Implementation of Pinkas Partial Matching (PM)-Semi Honest Protocol Using Mixed Multiplicative Homomorphic Encryption (MMH) For Location Based Services (LBS)

Levent Ertaul[†], and Saleha Shakoor[††]

[†]Faculty of Mathematics & Computer Science, California State University, Hayward, (510) 885-3356 USA
[††]School of Mathematics & Computer Science, California State University, Hayward USA

Summary
In the recent years smart phones have dramatically changed the face of the earth and the way the world operates by emerging itself as a pocket of technology. More then 50% of the smart phones owners use Location Based Services (LBS) of some kind everyday. The core component of the LBS is the proximity testing of users which determines if two mobile users are in vicinity to each other without claiming them to disclose their exact locations. In this paper we have worked on implementing Pinkas Partial Matching(PM)-Semi-Honest protocol using Mixed Multiplicative Homomorphic (MMH) encryption techniques that has the support of private proximity testing by utilizing location tags. This paper provides a practical assessment of proximity testing for LBS by implementing Partial Matching (PM)-Semi-honest protocol with MMH encryption. Android platform has been used for the implementation purposes of this protocol.

Key words:
Private-Set Matching & Intersection, Location Tags, MMH encryption, Location Based Services, PM-Semi-Honest Protocol.

1. Introduction

With the ongoing move of smart phones towards near-ubiquity, much of society has come to take these do-all devices for granted. According to one survey conducted by the "bitrebels.com", more then 80% of the cell phone owners check their smart phones within 15 minutes of waking up in the morning giving us a effulgent picture of how much everyday endeavors are influenced by this little device. The most significant feature of this device is Global Positioning System (GPS)[23] tracking that have virtually changed the concept of our society making concept of access and information available on the go at anytime and place. It embodies itself as pocket search engine empowering users to submit queries and receive their answers for any information they may desire or need. Such information can be gathered with respect to time and position by use of LBS [1]. Thus, one can define Location Based Services as-
"*Location Based Services are information services accessible with mobile devices through the mobile network and utilizing the ability to make use of the location of the mobile device*" [2]
There exist a number of LBS that includes Google Latitude, Foursquare, Loopt, Facebook places, and a large number of smart phone applications [4], [5]. One of the issues in LBS is the privacy which has been in debate for the recent years. Research and studies are being conducted to provide better privacy& security with this application. The primary challenge is that some of the people would dislike revealing their location information even to their closest friends at all times, yet allowing others to know some of the time [5].
This paper proceeds with the discussion of Privacy & Security concerns related to LBS in Section 2 followed by Proximity Services and Location tags in Section 3 and 4 respectively. Then Private Proximity Testing and Private Set Matching are discussed in section 5 and 6. Then PM-Semi Honest Protocol algorithm is explained in section 7 succeeded by the discussion of MMH encryption in section 8. The discussion is concluded at the end with the implementation of PM-Semi Honest Protocol with MMH encryption followed by the performance analysis of the implementation in section 9.

2. Privacy and Security Concerns in LBS

The subject of privacy is often addressed by the concern that how much of the sensitive information is to be kept secret in any application [7]. A substantial amount of privacy concern with the employment of LBS is the delivery of user precise location information with the un-trusted third parties. This concern applies to proximity services as well [1]. One of the biggest concerns is that it can be possible to compile a very detailed picture of someone's movements if they are carrying a wireless device that communicates its location to network operators [6]. LBS providers must alleviate consumer privacy fears by implementing secure network and encryption technologies to curb illegal activity [6], [16]. In general, privacy-preserving systems for LBS services are expected

Manuscript received March 5, 2015
Manuscript revised March 20, 2015

to satisfy **Location, identity & tracking protection**. **Mutual authentication** for security purposes should be implemented to prevent spoofing attacks.

3. Proximity Services

Proximity based services are a unique class of Location Based Services. These services inform users when they are within a certain distance of other people, businesses, or other things [10], [14]. Proximity testing is asymmetric which means one party will learn if the other party is nearby whereas the other party learns nothing [14]. In our paper we have demonstrated that it is indeed reasonable to deliver location functionality in a secure manner. In other words if the friends are nearby then they will be notified of their presence in the vicinity. No other information regarding their location or position will be revealed to either party.

Let us consider an application of proximity testing, keeping in mind that different applications require different proximity granularity [1].

- Alice and Bob who happened to be friends and wants to be serendipitously notified if they are in same shopping mall so that they can spend pleasant time together. Alternatively, Alice and Bob first meet online, but later decide to meet in person at a coffee shop. Alice arrives first and she wants to be notified when Bob arrives [1].
- Alice would like to get dinner with her friend Bob who travels a lot and wants to know if he is in town. Here the proximity granularity is a wide geographic area [1].
- Bob, a student lands at his college airport and wants to check if anyone from his college is currently at the airport so that he can share a ride with him or her to campus [1].
- Alice is a manager who wants to automatically record who is present at her daily meetings. However, her employees do not want their location tracked. Privacy preserving proximity testing over this well organized group allows satisfying both requirements [1].

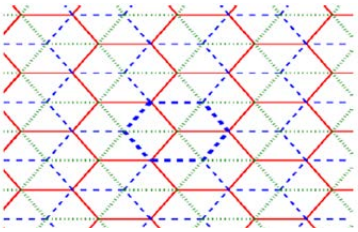

Figure 1. Three overlapping hexagonal grids. A blue grid cell is highlighted

4. Location Tags

A location tag is a secret consociated with a point in space and time. It is an assembly of location features deduced from (mainly electromagnetic) signals present in the physical environment. The matching function can be based on Hamming distance, set distance, etc [1].

4.1 Properties of Location Tags

The key properties of the location Tags are:

- *Unpredictability*- It should not be possible to generate matching tags unless and until nearby
- *Reproducibility*- Two devices at same place & time produce matching tags (not necessarily identical) [17].

Location tags provide a different model for proximity testing. The main advantage is that since the location tags of the two parties need to match, spoofing the location is no longer possible, which stops online brute force attacks [1]. The main disadvantage is that users no longer have control over the granularity of proximity: the notion of neighborhood is now entirely dependent on the type of location tag considered [1], [17], [12].

4.2 Construction of Location Tags

Several possible ways to extract location tags are [1]:

- *WiFi broadcast packets* comprises of a variety of different protocols (e.g. source and destination IP addresses, sequence numbers of packets, and precise timing information all offer varying degrees of entropy) offering a rich potential for extracting location tags information.
- *WiFi Access point IDs*: These points constantly advertise their presence using a combination of SSID and MAC Address which is already used by Skyhook for creating a database of ID-location mappings via “wardriving” [1].
- Bluetooth. Just like WiFi IDs, Bluetooth IDs are unique to each device giving it an advantage over WiFi IDs of almost always being associated with mobile rather than fixed devices, making Bluetooth-based location tags more time variable. The only limitation is the range making it a poor source of location tags [1].

- *GPS* works by timing signals sent by a number of orbital satellites. GPS consists of several signals, including "P(Y)" (encrypted precision code) and "M-code" (military) [1].
- *GSM*: Cellular towers that are in range in any one time seem like a promising location tag in the future due to its space- and time varying characteristics such as signal strength.
- *Audio:* In certain restricted conditions audio can serve as extractor for location for example music playing in background of coffee shop or a conversation in the meeting room.
- *Atmospheric gases:* [1] the cell-phone-as-sensor project has experimented with CO, NOx and temperature sensors on cell phones and hopes this will become standard on smart phones so that it can be used for real-time pollution mapping and other applications. If these sensors ever become main stream, they are another potential source of location tags.

5. Proximity testing using location tags

Private set intersection is a well studied problem and an elegant solution was given by Freedman, Nissim and Pinkas [10]. The protocol of Freedman, Nissim and Pinkas makes use of a homomorphic encryption Scheme E that anyone can encrypt and Alice can decrypt. At the end of the following protocol, Alice learns $|A \cap B|$ and Bob learns nothing [1].

So the protocol would work as follows:

1. Alice generates a polynomial p from her set of location tags.
2. Alice then sends the encrypted polynomial coefficients $E(p)$ to Bob.
3. Bob the calculates his own polynomial $p(b)$ with his location tags and then encrypts it as $E(p(b))$.
4. Then Bob picks a random number r on $E(p)$ and computes $E(r(p(b))$ using Homomorphic encryption.
5. Then Bob sends Alice the permutation of encryptions computed in earlier step.

Alice then decrypts it and outputs the nonzero decryptions as intersection of A and B.

6. Private Set Matching and Set Intersection

The protocol follows the following basic structure. Alice defines a polynomial P whose roots are her inputs [10]:

$$P(y) = (x_1 - y)(x_2 - y) \ldots\ldots (x_k - y) = \sum_{u=0}^{kc} \alpha^u y^u$$

She sends to Bob homomorphic encryptions of the coefficients of this polynomial. Bob uses the homomorphic properties of the encryption system to calculate the polynomial at each of his inputs. He then multiplies each result by a fresh random number r to get an intermediate result, and adds to it an encryption of the value of his input, i.e., Server bob computes $Enc(r \cdot P(y) + y)$. Therefore, for each of the elements in the intersection of the two parties' inputs, the result of this computation is the value of the corresponding element, whereas for all other values the result is random [10].

7. Pinkas Partial Matching (PM)-Semi-Honest Protocol using Location tags

Working of this protocol will be explained with demonstration of an example. Assume the following [20]:

Let Input: Client's input(Alice) is a set $X=\{x1, \ldots, xk\}$, Server's input (Bob) is a set $Y=\{y1, \ldots, yk\}$.

The elements in the input sets are location tags taken from a domain of size N.

1. Alice performs the following operations:

(a) Alice selects the secret-key parameters for a semantically-secure homomorphic encryption scheme, and publishes its public keys and parameters. The plaintexts are in a field that contains representations of the N elements of the input domain.

(b) Alice uses interpolation to compute the coefficients of the polynomial

$P(y) = (x_1-y)*(x_2-y)* \ldots *(x_k-y)=a_k*y\hat{}k+ \ldots +a_1*y+a_0$, of degree k, with roots $x_1, \ldots, x_k$.

(c) Alice encrypts each of the $(k+1)$ coefficients by the semantically-secure homomorphic encryption scheme and sends to Server Bob the resulting set of ciphertexts, $\{Enc(a_0), \ldots, Enc(a_k)\}$.

Then, this information is communicated to Server Bob.

2. Bob performs the following for every y in Y,

(a) He uses the homomorphic properties to evaluate the encrypted polynomial at y. That is, Server Bob computes $Enc(P(y))=Enc(a_k*y\hat{}_k+ \ldots +a_1*y+a_0)$.

(b) Bob selects a random value r and computes $Enc(r*P(y))$.

(c) Bob randomly permutes this set of k ciphertexts.

Then Bob sends the result back to the Alice.

Alice decrypts all k ciphertexts received. She locally outputs all values x in X for which there is a corresponding decrypted value.

In the original paper [10] the encryption scheme recommended was Pailler, developed by Pascal Paillier in Public-Key Cryptosystems Based on Composite Degree Residuosity Classes, and is semantically secure. It is an additively probabilistic and asymmetric algorithm which is homomorphic and self-binding. It states that given composite n and integer z, it is hard to determine if y exists such that $z = y^n \pmod{n^2}$. In Paillier you choose two large prime numbers p and q which are randomly and independently selected of each other such that $gcd(pq, (p-1)(q-1))=1$ This property is assured if both prime numbers are of equal length [18]. You then compute $n=pq$ and $\lambda=lcmp(p-1,q-1)$. Then g is computed by taking a random number such that $g \in \hat{Z}_n^2$ such that $L(g^\lambda \bmod n^2)$ is invertible modulo n where $L = x = \frac{u-1}{n}$ where n and g are public encryption keys and p and $q(\lambda)$ are private decryption keys respectively that is

$$\text{Encryption Keys}(x,r) = g^m \bmod n^2$$

$$\text{Decryption Keys } (y) = = \frac{L(y^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)}$$

As far as performance analysis of Pailler encryption scheme is concerned, Pailler showed poor performance results [19].The encryption time of Paillier depicted huge time difference compared to RSA [22] and ElGamal [23]. RSA showed better performance over ElGamal and Paillier in term of encryption time and ElGamal showed better performance over RSA and Paillier in term of decryption time [19]. When throughput analysis was performed RSA showed better throughput over ElGamal and Paillier in encryption process and ElGamal showed better throughput over RSA and Paillier in decryption process [19]. Paillier showed worst result in encrypted file size with exponential increase of the input file size. The best result was shown by RSA. ElGamal, RSA. Paillier showed same exponential increase in decrypted file size with increase in input file size [19]. The main reason was the complex computation involved in the encryption and decryption process. Due to the complexity of Pailler we needed to find the encryption scheme that was simple and required less computation with better performance and required security and hence we opt for MMH.

8. Mixed Multiplicative Homomorphism (MMH)

In Homomorphic encryption schemes operations can be performed on the encrypted or cipher text just as they are performed on the plain text. A Homomorphic encryption scheme depicts the property of additive, multiplicative and mixed multiplicative homomorphism. For the implementation of PM-Semi honest protocol we have used mixed multiplicative homomorphism. In mixed multiplicative homomorphism, decrypting the product of one ciphertext plaintext is same as multiplication of two plaintext, represented as $E(x*y) = E(x) * y$. [8], [9], [11], [12]. Crypto system implements MMH using large prime numbers p and q that are secret for calculating a public parameter n such that $n=p*q$. The set of original plaintexts is in $Z_p = \{ x | x <= p \}$ and the set of ciphertexts is in $Zn = \{ x | x \}$ whereas $Q_p = \{ a | a \notin Z_p \}$ has the set of encryption clues.

The encryption process chooses plain text $'x' \in Z_p$ and a random number $'a'$ in Q_p such that $x = a \bmod p$ and ciphertext y is calculated as $y = Ep(x) = a \bmod n$. In the decryption process plaintext is recovered as $x = Dp(y) = y \bmod p$, where p and n are private public keys respectively. MMH has the property of additive, multiplicative and mixed multiplicative homomorphism.

Here is a simple example for demonstration. Let the cryptosystem select $p=13, q=5$ from which we calculate $n=p*q= 13*5=65$. Let $x_1=3$ and $x_2=7$. Now x should be in $a \bmod p$ that is $x_1=3=a \bmod p \rightarrow 3=94 \bmod 13$ from which we get $a=94$ and now $Ep(x_1)= a \bmod n = 94 \bmod 65 = 29$.Similarly $x_2= 7 = a \bmod p \rightarrow 7=111 \bmod 13$ from which we have $a=111$ and then $Ep(x_2)= a \bmod n=111 \bmod 65=46$. Now $Ep(x_1) * x_2 = 29 * 7 =203 \bmod 65 = 8$. If we now decrypt it that is $Dp(8) = 8 \bmod 13 = 8$, which is same as the plaintext $x_1 * x_2 = 21 \bmod 13 = 8$. This shows that mixed multiplicative homomorphism can be performed on the ciphertext, as if performed on the plaintext [13].This example shows simplicity of MMH giving it an advantage over Paillier's complex encryption and decryption schemes.

9. Implementation of Pinkas PM-semi-honest protocol with MMH encryption

We have implemented the PM-Semi Honest protocol in Android studio with Geny Motion. Client Server model has been used to implement this protocol with the help of JAVA language [24]. Single emulator is run for both Alice & Bob users to speed up the simulation time.

The detailed specification of the implementation is shown in the following table.

In order to present a better insight into the implementation of the protocol I would demystify it with an example.

Let's assume Alice is in the vicinity of Bob. Now Alice wants to be notified of Bob's presence in the neighborhood. Alice (Client) will generate its input set called roots from a domain of size N. In our implementation its $X=\{x_1,x_2,\ldots.x_k\}$ of client Alice. The number of the roots generated for the location tags can be user choice. These roots represent the location tags associated with the user. These roots are generated randomly by the function *Random Generator* from Math Library of Java since

location tags are difficult to be calculated with the present hardware available.

Table 1: Specification of Implementation

Platform	Intel® Core (TM) i7-2720QM CPU@2.20Ghz Installed RAM 4 GB(3.24 usable)
Operating System	Windows 7
Programming Language	Java
Run time Environment	JRE 8 [24]
Network Model	TCP/IP Client/ Server Model
IDE/Other Platforms	Eclipse/Android Studio with Geny Motion
Crypto Library	Java.Security [24]
Encryption mode	MMH
Key size & Security parameters	Private/Public key selected by user .Options include 128,256,512 & 1024 bits
Input set	User selects the number of roots (Location Tags). Options include 2,4,8,16 and 32 roots
Domain	2 cities (Freemont, Hayward)

The size of each root will add up to be total size of ***128*** bits in our implementation. For example if ***2*** roots are selected then size of each of these roots will be ***64*** making total of ***128*** bits. In this way we have worked with maximum ***32*** roots, each of ***4*** bits for the total of ***128*** bits. For the sake of simplicity in our example we select 2 roots for Alice i.e. ***X= {2, 4}*** roots which are randomly breaded as mentioned earlier. Remember ***2*** and ***4*** each are ***64*** bits giving a total of ***128*** bit location tags. In our case we have generated the roots as follows:

```
SecureRandom random = new SecureRandom();
byte bytes[] = new byte[16]; // 128 bits are converted
to 16 bytes;
            for (int i = 0; i < rootsNum; i++) {
                  random.nextBytes(bytes);
                  BigInteger p = new
BigInteger(totalBits/rootsNum, random);
Random rand = new Random();
randP[i] = rand.nextInt(29 -11) + 11;
                  list.add(p[i]);
}
```

Now Alice will employ these roots to generate a polynomial which is $P(y)=(2-y)(4-y)= 8-6y+y^2$. She will now encrypt this polynomial with the homomorphic scheme that is MMH. Coefficients ***8, 6, 1*** will be encrypted as follows:

P=13 (Private key), q=5 ,n=p*q =65 (Public key)
a_0 =8 , a_1=-6, a_2 =1
For ***a_0*** we calculated ***a =99*** using ***a_0=a*mod p***
$Enc(a_0)$=99*mod 65=34, sly for
$Enc(a_1)$ =45 , $Enc(a_2)$=40

```
private EncryptedPolynomial
getPolynomial(BigInteger []c) {
            EncryptedPolynomial poly = null;
            try {
      poly = new EncryptedPolynomial(c, pubkey);
            } catch (BigIntegerClassNotValid
bigIntegerClassNotValid) {

bigIntegerClassNotValid.printStackTrace();
            }
            return poly;
      }
```

Now these encrypted coefficients will be sent to Server Bob. Bob will use these encrypted coefficients to generate polynomial at its location tags. Since Bob is in the neighborhood it will also yield the matching roots from domain of size ***N*** i.e. ***Y ={ 2,4}***. A random number r will be generated and following will be executed:
$P(y)=(2-y)(4-y)= 8-6y+y^2$, Let r=1 for simplicity
$r*Enc(a_2)*y^2 - r*Enc(a_1)y + r*Enc(a_0)$

These computations are done for all values of ***Y*** that's ***2, 4*** by using MMH homomorphic properties.

```
EncryptedPolynomial p = new
EncryptedPolynomial(this);
EncryptedCoefficient[] p_coefficients = new
EncryptedCoefficient[this.coefficients.length*2-1];
for (int i=1; i <= this.coefficients.length; i++) {
  for (int j=1; j <= plain_coefficients.length; j++) {
    int l = i+j-2;
    EncryptedCoefficient p2
=this.coefficients[i-1].multiply(plain_coefficients[j-1
]);
          if (p_coefficients[l] == null) {
                p_coefficients[l] = p2;
                      }
                else {
      p_coefficients[l] = p_coefficients[l].add(p2);
                      }
                }
}
```

After encryption random permutation is performed on K ciphertexts and is sent to client Alice.

```
public String[] permute(String[] data)
{
        for (int i=0; i<data.length; i++) {
            int randomPosition =
rgen.nextInt(data.length);
            System.out.println("Random value
generated:"+randomPosition);
            String temp = permutation[i];
            permutation[i] =
permutation[randomPosition];
            permutation[randomPosition] = temp;
        }
        return permutation;
}
```

Alice will execute depermutation function on these ciphertexts and will then place its input to this equation. If the sum is ***0*** they are near by else they are far. Since in our case they are near so output is ***0.***

One thing worth mentioning is that our private and public/private keys are of ***128/256/512/1024*** bits which are selected by the user at the time of simulation. Following snapshot will better clarify our implementation work.

Figure 2 depicts Alice emulator side where user can choose its location, size of the key and number of roots for location tags.

Figure 2 Alice

Figure 3 is the Bob side. Its roots and key sizes are automatically adjusted according to Alice inputs and devising it on Bob side was not a necessity.

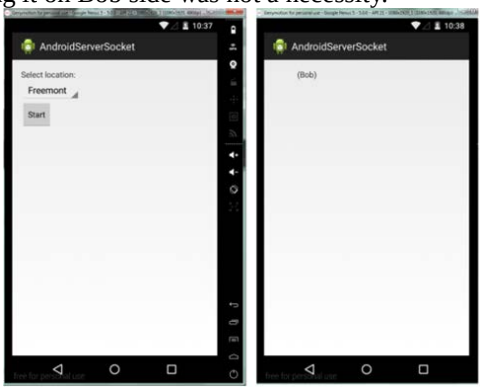

Figure 3 Bob

Two cases are simulated with this protocol which we will now be discussing. We have used two cities for location selection. One is Freemont and other is Hayward.

Case 1: When Alice & Bob both are in Freemont

Key Size used=***1024*** bits, Number of roots which are inputs to generate the polynomial (Location tags) used=***32*** (each roots is of 4 bits giving a total of ***128 bits=32*4***)

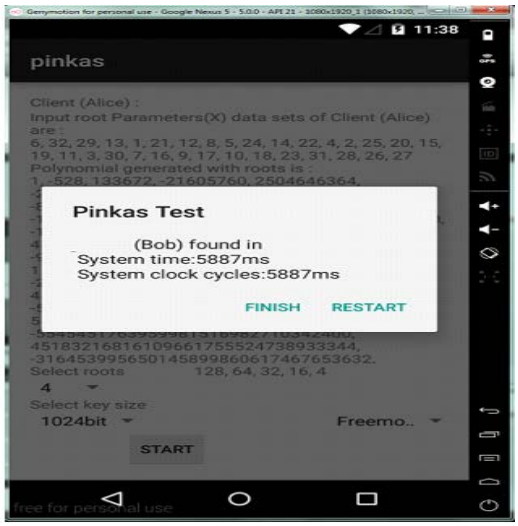

Figure 4 Alice results

In Figure 4 the result come backs to Alice when she enters the start button from Figure 2. It took almost ***5887ms*** to generate the results. This simulation time is recorded for the performance measurement later.

Figure 5 Alice results

In Figure 5 we see that the decrypted sum calculated out to be ***zero*** by the protocol using MMH implying Alice and

Bob are in the vicinity.

Figure 6 Bob results

Figure 6 shows the encrypted coefficients received from Alice by Bob from MMH which it will use for its polynomial generation. Bob will also calculate this polynomial with MMH encryption. Due to limited space we are displaying few coefficients only.

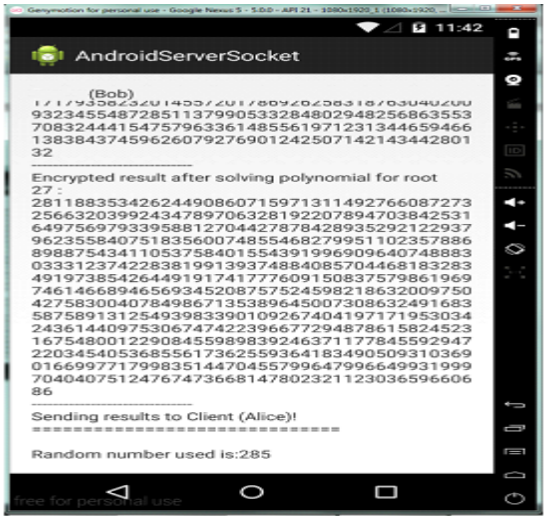

Figure 7 Bob Results

Figure 7 shows Bob encrypted polynomial for its location tags. Only few have been displayed in this screen.

Case 2: When Alice is in Freemont & Bob is in Hayward
Key Size=***512*** bits, Number of roots (Location tags) used= 8 (each root is of ***16*** bits giving total of ***128 bits=8*16***)

Figure 8 is the result when Alice & Bob are at different locations. It took almost ***396ms*** to complete the simulation and generating the output that Bob is not in the neighborhood.

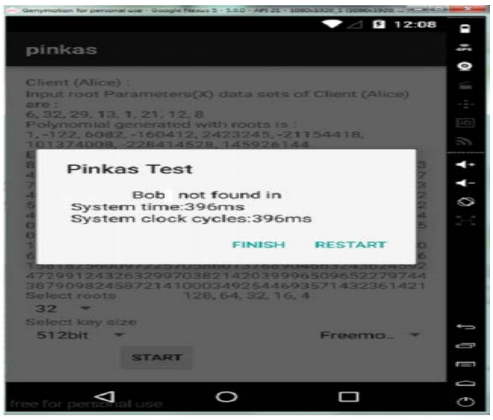

Figure 8 Alice results

Figure 9 Alice Results

Figure 9 show the decrypted sum of the protocol which resulted in a ***non- zero*** value and hence it concluded that Bob is not near by.
Figure 10 shows encrypted coefficients received by Alice in this case 2. These are only few coefficients shown out of many.

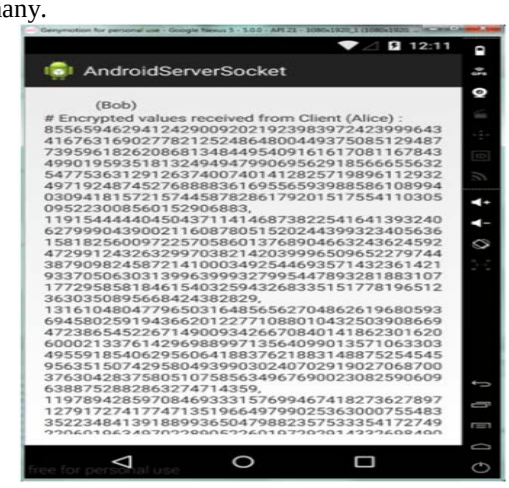

Figure 10 Bob Results

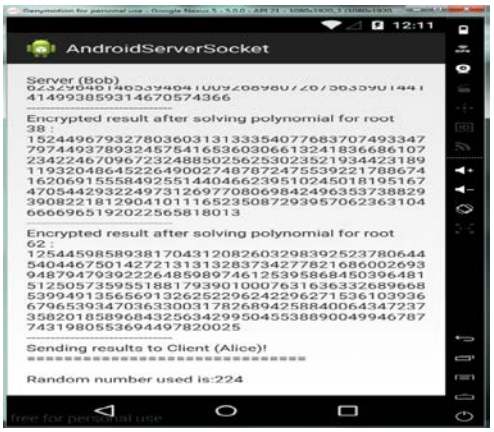

Figure 11 Bob Results

Figure 11 show the few calculated polynomials of Bob for its location in case number 2.

9. Performance Measurements

When the protocol was run on Android Studio with Geny motion with key size of ***1024*** bits and ***32*** roots the average time it took Alice to measure the proximity of Bob was ***6 seconds***. As the number of roots starts to decrease the performance was getting better. While measuring performance we are assuming that keys are already generated that is we are not measuring the time of generation of the keys. As a result we observed that there is not a significant change in the simulation time. This shows that the protocol is key size independent. The only factor affecting its performance is the number of roots/ location tags. To compare the performance of the MMH encryption 2 graphs have been plotted. One is between different key sizes with same number of roots & other one plot with different number of roots but same key sizes.

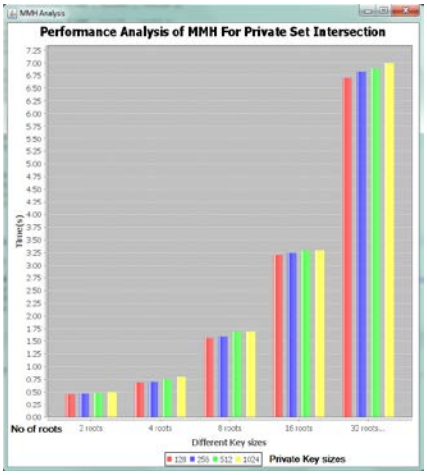

Figure 12 Comparison of Different Key sizes with same number of roots

From figure 12 we analyses that with change in size of encryption key there is no effect on the time taken to simulate the protocol. But in Figure 13 we see huge effect on simulation time when the numbers of location tags are increased. we have tried running the same protocol in eclipse with its android sdk and my experience tells me that simulation gets pretty faster if Android studio is used with Geny motion instead [15]. There is almost a difference of ***2-5*** seconds in eclipse with android sdk, depending on the roots used, compared to same protocol being run in the Android studio with Geny Motion. Table 2 shows the simulation results of the protocol with different key sizes and roots

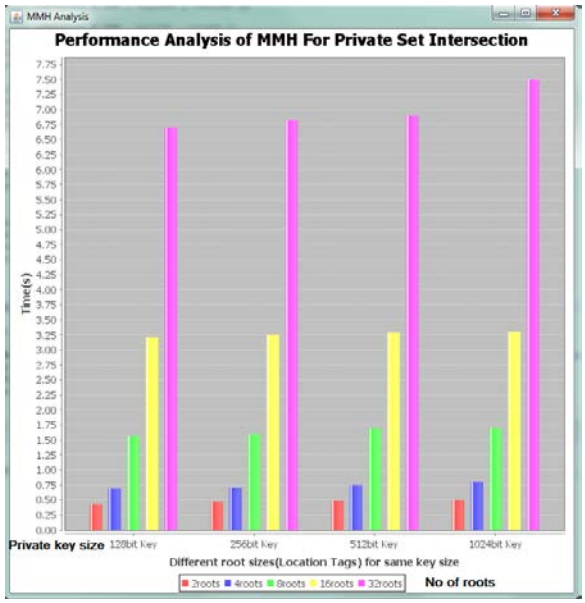

Figure 13 Comparison of different number of roots for same key size.

In Table 2 the column “Time taken” is the time it took to start the protocol with MMH encryption and displaying the results on the screen. It is the system time that Figure 4 and Figure 8 are displaying in their screen shots by the name “System time”. Similarly by choosing different key sizes and root sizes we have measured the time taken in each case. The rows of the “Time Taken” are filled by running the simulation 10-15 times and then taking average of these values to fill up the rows of the table. Table also lists columns of different encryption key sizes and number of roots used to measure the performance.

10. Conclusions

Location privacy has evolved itself as a growing concern among smart phone users. In this paper we implemented MMH encryption to simulate Pinkas PM-semi-honest protocol and have measured its performance. Earlier this

protocol was implemented with complex and huge computation encryption scheme like Paillier. But in this paper we have proved that MMH encryption works for Pinkas protocol assuring that it could be a good candidate for the implemetation of Pinkas PM-Semi-Honest protocol with far less computation overhead.

Table 2: Simulation Results with different key sizes and number of roots

Number of roots	key size	Time taken
2	128	460ms
	256	469ms
	512	485ms
	1024	500ms
4	128	690ms
	256	699ms
	512	750ms
	1024	800ms
8	128	1570ms
	256	1600ms
	512	1690ms
	1024	1700ms
16	128	3210ms
	256	3250ms
	512	3290ms
	1024	3300ms
32	128	6700ms
	256	6820ms
	512	6900ms
	1024	7000ms

We ran MMH encryption with diffetent root sizes and encryption key sizes and have successfully attained results with it. The size of the encryption key has no signficant change in simulation time of Pinkas PM-Semi honest protocol with MMH encryption. The only factor that effeted its performance were the number of the roots used to create polynomial location tags demonstrated a proportial increase in time. Our implementation results has encouraged us the use of MMH encryption for real time applicaions. The reduction in computation overhead with MMH encryption embolds its use for mobile applications where limited battery life is a key factor in determining the performance of the protocols. More computation means more power consumption resulting in less battery life and hence MMH encryption inconvertibly becomes the favorite. However, MMH encounters disadvantages like known plaintext attacks and integrity attacks as decryption is performed in ***modulo p***, any unencrypted number $x < p$ will be deciphered as itself [17].

References

[1] A.Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. “Location privacy via private proximity testing”, Network and Distributed System Security Symposium, NDSS, 2011.

[2] Virrantaus, K., Markkula, J., Garmash, A., Terziyan, Y.V., 2001.“ Developing GIS-Supported Location-Based Services”, In: Proc. of WGIS’2001 – First International Workshop on Web Geographical Information Systems., Kyoto, Japan. 423–432, 2001.

[3] Open Geospatial Consortium (OGC), Open LocationServices.www.nttdocomo.com/corebiz/network/index.html Internet information on mobile networks, 2005.

[4] S.Mascetti, Claudio Bettini, Dario Freni, X. Sean Wang, X. Sean Wang, Sushil Jajodia. “ Privacy-Aware Proximity Based Services”, Tenth International Conference on Mobile Data Management: Systems, Services and Middleware, 2009.

[5] J.Dam Nielsen, Jakob Illeborg Pagter, and Michael Bladt Stausholm. “ Location Privacy via Actively Secure Private Proximity Testing”, Fourth International Workshop on SECurity and SOCial Networking, Lugano, 19 March 2012.

[6] C. Steinfield. “ The Development of Location Based Services in Mobile Commerce”, Technology Management for Reshaping the World. Portland International Conference, 2004.

[7] L. Barkuus, and Anind Dey. “ Location-Based Services for Mobile Telephony: a Study of Users’ Privacy Concerns”, 9TH IFIP TC13 International Conference on Human-Computer Interaction, 2003.

[8] J. Domingo-Ferrer. “A Provably Secure Additive and Multiplicative Privacy Homomorphism”. Information Security Conference, LNCS 2433, pp 471–483, January 2002.

[9] J. Domingo-Ferrer and J. Herrera-Joancomarti. “A privacy homomorphism allowing field operations on encrypted data”, I Jornades de Matematica Discreta i Algorismica, Universitat Politecnica de Catalunya, March 1998.

[10] M. Freedman, K. Nissim, and B. Pinkas. “Efficient private matching and set intersection”, In Proc. of Eurocrypt’ 04, pages 1–19. Springer-Verlag, 2004.

[11] Hyungjick Lee, Jim Alves-Foss,Scott Harrison, “ The use of Encrypted Functions for Mobile Agent Security”,Proceedings of the 37th Hawaii International Conference on System Sciences – 2004.

[12] J. Domingo-Ferrer, “A new Privacy Homomorphism and Applications”, Elsevier North-Holland, Inc, 1996.
[13] Gorti VNKV Subba Rao. “Secured Data Comparison in Bioinformatics using Homomorphic Encryption Scheme”, Global Journal of Computer Science and Technology. Volume 1.2, 2009 Sep
[14] Mike Hamburg, Joint work with Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Dan Boneh “Location Services with Built--In Privacy,” 2011
[15] http://www.quora.com/Which-is-better-Eclipse-or-Android-Studio
[16] http://www.sitepoint.com/improved-android-emulation-genymotion/
[17] Sachin Upadhye1, P.G. Khot2. “Homomorphic Encryption Scheme & Its Application for Mobile Agent Security”, International Journal of P2P Network Trends and Technology(IJPTT), Dec 2013
[18] Alexander Lange ,”An Overview of Homomorphic Encryption” , PKC 2012, 15th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA
[19] Shahzadi Farah, M. Younas Javed, Azra Shamim and Tabassam Nawaz.” An experimental study on Performance Evaluation of Asymmetric Encryption Algorithms”, Proceedings of the 3rd European Conference of Computer Science (ECCS '12)
[20] http://www.google.com/patents/US20060245587
[21] Elgamal, Taher (1985). "A Public key Cryptosystem and A Signature Scheme based on discrete Logarithms". IEEE Transactions on Information Theory
[22] Ron Rivest, Adi Shamir and Leonard Adleman, https://tools.ietf.org/html/rfc2437
[23] http://www.gps.gov/
[24] http://www.java.com/en/

Levent Ertaul received B.Sc. from Anatolia University Turkey, in 1984, M.Sc. from Hacettepe University, Turkey, in 1987, and PhD degree from Sussex University, UK, in 1994. After working as an assistant professor (from 1994) in the Dept. of Electrical & Electronics Engineering, Hacettepe University, he moved to California State University, East Bay in 2002. He is currently a full time Professor at California State University Eastbay, USA in the department of Math & Computer Science. He is actively involved in security projects nationally and internationally. His current research interests are Wireless Security, Ad Hoc Security, Security in WSNs and Cryptography. He has numerous publications in security issues.

Saleha Shakoor is a graduate student in California State University, East Bay, CA, USA