An Efficient Private Matching and Set Intersection Protocol: Implementation PM-Malicious Server

G. Saldamli, L. Ertaul, K. Dholakia, U. Sanikommu

Abstract – ***The Private Set Intersection (PSI) is a privacy based service in which the service adaptation depends on the intersection set of two clients each of which has a large domain. The goal is to know only the common part and not to disclose unnecessary information. With the growing demand in privacy based services, in this paper we show how to implement the well-known Pinkas algorithm for malicious server. As smart phones are becoming extremely popular and most trusted computing device so we implement Pinkas protocol using Java language to make an Android application on Eclipse IDE interface. We would also show how the protocol would perform for different set of inputs.***

I. INTRODUCTION

The enormous growth of computers and users and their interaction of sharing data have led to a number of concerns like protecting data, preventing resources, guarantee authentication etc. Our paper deals with one of these concerns, which is sharing of data between two parties without disclosing additional information but only to share the information, which is common between the two. Not only this but to find a secure method to do the same. In a private set intersection protocol a client 'C' and a server 'S' jointly work out the intersection of their private sets in a way that they find 'C∩S' at the end. If the client learns the intersection and the server learns nothing, this method is called one way Private Set Intersection. If both learn about the intersection then it is known as mutual Private Set Intersection. [1][2][3]

This type of protocol is necessary when two parties don't trust each other and don't want to disclose all of their information. One such example can be two pharmaceutical companies want to know if any of their customers have been using illegal prescriptions but don't want to share all their customer information. Then they can use this protocol to see if there is any common customer who is committing the fraud. There can be many other implementation of this protocol like online dating service, it can be used by government for security purposes, or any other organization which want their data to be private and share data at the same time.

G. Saldamli is with Computer Engineering Department at San Jose State University, San Jose, CA, USA
L.Ertaul is with Department of Computer Science at California State University, East Bay, Hayward, CA, USA
K.Dholakia is a student at California State University, East Bay, Hayward, CA, USA
U.sanikommu is a student at California State University, East Bay, Hayward, CA, USA

In this paper we use Private Matching (PM) protocol [2] which is based on homomorphic encryption and randomization of the results. This protocol is efficient for both semi honest parties and malicious adversaries. The protocol is flexible and can also calculate the intersection size of two sets. We can modify it for some function to be performed on the intersection set. The protocol can also be extended for multi-parity intersection or for several instances. [9][7][12][13][14][15][16][17]

The rest of the paper is organized as follows. **Section 2** describes some of the related work and preliminary terms. **Section** 3 explains PSI based issues. **Section 4** illuminates the protocol for malicious adversaries. **Section 5** illustrates and implements our work on Private Matching for malicious servers. **Section 6** shows screenshots of our implementation and finally in **Section 7** we give a conclusion with a short discussion.

II. RELATED WORK AND PRELIMINARIES

In this section first we describe some of the related work and then we describe some basic terms.

Some of the related work are: Private Set Equality test is another implementation of private matching which is securely computed through gates but it had overheads which you can see in [4][5]. Disjointness function used DISJ (a, b) [10] where it returns 1 if the intersection was null. A lot of research has been done to study the complexity of this function. Conclusion was that even excluding the privacy factor the complexity of private matching will be at least proportional to the input size. One solution to this was to approximate the intersection size by private approximation protocol [20].

Before we explain the protocol some of the terms that you should know about are: Private Matching of client 'C' and server 'S' where size of client set is kc from some large domain N and size of server set is ks from the same domain. Let us suppose C input is $X = (x_1 \ldots x_{kC})$ and S input is $Y = (y_1 \ldots y_{kS})$, Then C learns

$$X \cap Y = \{x_u \mid \exists\, v;\ x_v = y_v\}$$

But S learns nothing about the intersection. Some other variants used by PM are (i) Private Cardinality matching PM_C which helps the client to find out exactly what it shares with the server. (ii) Private threshold matching PM_t which helps the client to know whether the intersection was above or below a certain threshold. A threshold that was predicated before the intersection. (iii) Based on the output of PM_C and PM_t we can make arbitrary private matching protocol.

To know about the malicious party first you should know what is a semi honest model as it is derived from it.

Semi-honest model is the one in which both client and server act according to the protocol. The client privacy is indistinguishable, in more general term it means that for different input sets of client the server wouldn't know the difference because the server doesn't get any output anyway. Also the client wouldn't get any false information than what is it supposed to get. We can make sure of this by introducing a trusted third party who ensures authentication and integrity. For malicious adversary model there can be three cases: (i) the client or server can refuse to take part in the protocol, (ii) or it can modify its input to a false value, (iii) else it can abort the protocol prematurely. Our main concern for this protocol implementation is to ensure its security rather than its privacy. We can enforce the privacy and correctness issues of the protocol and is explained in [6]. The protocol is also limited when only one party is honest either the client or the server.

We can use either of the homomorphic encryption schemes - additive homomorphic or multiplicative homomorphic. In Additive homomorphic addition is done on cyphertext and on decryption it gives us the same result as the addition of all plaintext. Suppose we have Enc (m1) and Enc (m2) then their additive homomorphic encryption will be Enc (m1+m2). For multiplicative homomorphic encryption we multiply it with some constant from the same domain and on decryption it will give us the multiplication of all plain text. It can be represented as Enc (cm) where c is constant. Another homomorphic scheme is mixed multiplicative homomorphic encryption, this a multiplication of two large prime no p and q represented as m=p*q. We will discuss about this scheme in other section. [21][22][23]

III. PSI- SECURITY ISSUES AND CONCERNS

This section explains how security is maintained in Private Set Intersections. The privacy of both client and server are preserved because the client data is private as the encryption in PSI [18][19] is semantic and server won't be able to distinguish for two different input of client. The data of sever is private because for C operating in idle model and C* operating in real model won't be able to distinguish between its Y input for the server. The security of hashing based protocol [11] is also preserved for client and server as client still uses semantic encryption and key chosen is not dependent of client input. Hence sever can never know the X as neither the key nor set is disclosed. For server the privacy is preserved by using non-zero roots for the polynomial.

Another famous PSI protocol is oblivious transfer protocol [4] where the sender sends a part of its input to the client such that both the parties are protected hence the sender wouldn't know which part was transferred and the client wouldn't know which part it received. You should have noticed that the sender view is independent of client and this would guarantee the client even if server tires to cheat. The security of the sender when the client is semi-honest is preserved by S where output is indistinguishable from C(x1, x2). In case of intersection the client should learn the set of $X \cup Y$ but not the sets of these elements. We only need to make few changes in server if we want to add this functionality. Server has to compute Enc (rP(y) + 0^+). Then the client calculates the number of ciphertext from the server which then decrypts it to string 0+ and gives c. These are few schemes that help us maintain integrity and privacy.

IV. EXPLANATION- PM-MALICIOUS-SERVER

How to securely communicate the information in presence of malicious parties is the main goal of our paper. First we have to understand that either the server or the client any one can be malicious. We have different protocol for both the client and the server. Though our implementation result is only for the server side. What we should keep in mind is that the sever protocol is secure in random oracle model but the client protocol is analyzed in the standard model. Let us first start our discussion with the malicious client.

A. Malicious Client

For a malicious client we must keep in mind that the input of client set in ideal model can effectively simulated in his view of real model. The protocol can work with and without hashing technique. When we don't using hashing technique in protocol we enforce that the client sends the polynomial coefficients which has at least one non zero input, if it doesn't do so then it can't distinguish whether the input was present in the server side. In case of hashing technique we use the method of cut-and-choose [8] with B polynomials with each of degree M. Cut-and-choose method ensures the server that it used the same technique for hashing as it was agreed upon. Here the client chooses a key for the pseudo-random function and hashes the function and sends it to the server, it adds a lot of zero's so that the degree is M. These steps are repeated L times till L copies are created with each having a different key. Once the server gets the input he asks the client for L/2 copies, it does but without revealing the keys. The server then verifies each of the copies with the hash function to generate his own pseudo-random identities, and runs it for each of his polynomial. Then the server sends the result to client and it decrypts the L/2 set and compares it with own set to find the intersection.

B. Malicious Server

Like we mentioned above the protocol will first compute the polynomial of roots of the input set. It can be calculated by the following equation:

$$P(y) = (x_1-y)(x_2-y)\ldots.(x_{kc}-y) = a_0 + a_1y + \ldots. + a_ky_k \qquad (1)$$

Then it performs homomorphic encryption on each of the coefficients of the polynomial equation. The encryptions are then sent to the server. The server along with its input evaluates each of the polynomial using the homomorphic properties. The server then multiplies each result with a random number 'r' to get an intermediate result. Hence server computes Enc (r.P(y)) + y). So in the end we will have the inputs of the common elements along with the random number that were added. Else the intersection will just have the random number if there is no common input. One thing to notice is that there can be overhead which is exponential. For a balanced allocation function [7] we must perform mapping, where elements are thrown to bins of range size B which is selected by the client. We use balanced allocation hashing scheme in which the element is put into a bin which is less occupied. And the set we obtain is

$$(e,h) \leftarrow (Enc(r'.P(y) + s), H2\ (r'',y)) \quad (2)$$

A malicious server can play tricks on his input; it can give 'C' two different sets, it can modify the value of encryption of polynomial. We make few changes in the protocol to ensure security against malicious sever. If we force our server to follow a specific procedure then it can ensure integrity. The protocol ensures the security of the client. It does so assuming there are two servers: one for the real model S* and one for the ideal model S. If S sends it value to a trusted third party and if the output of S and S* match we know the server is not malicious. The figure below describes how the set (e, h) is computed on server side and later how it is decrypted by the client.

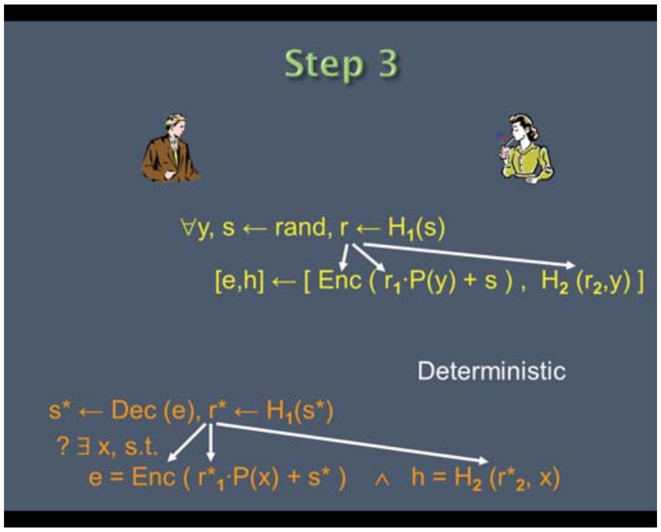
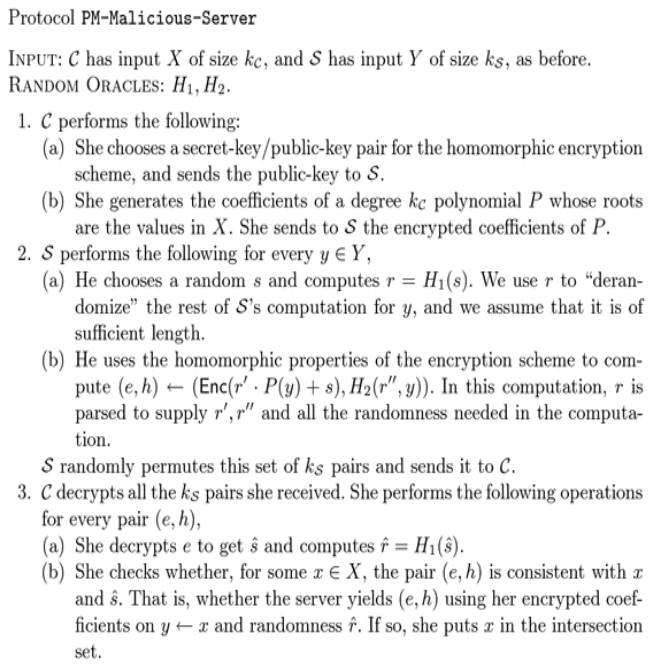

Step 3
∀y, s ← rand, r ← H1(s)
[e,h] ← [Enc (r1·P(y) + s) , H2 (r2,y)]
Deterministic
s* ← Dec (e), r* ← H1(s*)
? ∃ x, s.t.
e = Enc (r*1·P(x) + s*) ∧ h = H2 (r*2, x)

Figure1. Encryption and decryption scheme

There can also be a case where we can have both client and server as malicious. Here the client generates B bins as it did before and a polynomial is generated with degree M, here the L/2 copies are opened by the client and it recovers the S from it and follows the same procedure as PM-Malicious Server. But a question arises that are the malicious parties secure in standard model, as the proof we have are only for random oracle model. Also how secure and efficient is the protocol. The figure below describes the steps that we followed to implement this protocol.



Protocol **PM-Malicious-Server**

INPUT: $\mathcal{C}$ has input X of size $k_{\mathcal{C}}$, and $\mathcal{S}$ has input Y of size $k_{\mathcal{S}}$, as before.
RANDOM ORACLES: H_1, H_2.

1. $\mathcal{C}$ performs the following:
   (a) She chooses a secret-key/public-key pair for the homomorphic encryption scheme, and sends the public-key to $\mathcal{S}$.
   (b) She generates the coefficients of a degree $k_{\mathcal{C}}$ polynomial P whose roots are the values in X. She sends to $\mathcal{S}$ the encrypted coefficients of P.
2. $\mathcal{S}$ performs the following for every $y \in Y$,
   (a) He chooses a random s and computes $r = H_1(s)$. We use r to "derandomize" the rest of $\mathcal{S}$'s computation for y, and we assume that it is of sufficient length.
   (b) He uses the homomorphic properties of the encryption scheme to compute $(e,h) \leftarrow (\mathsf{Enc}(r' \cdot P(y) + s), H_2(r'', y))$. In this computation, r is parsed to supply r', r'' and all the randomness needed in the computation.

   $\mathcal{S}$ randomly permutes this set of $k_{\mathcal{S}}$ pairs and sends it to $\mathcal{C}$.
3. $\mathcal{C}$ decrypts all the $k_{\mathcal{S}}$ pairs she received. She performs the following operations for every pair (e,h),
   (a) She decrypts e to get $\hat{s}$ and computes $\hat{r} = H_1(\hat{s})$.
   (b) She checks whether, for some $x \in X$, the pair (e,h) is consistent with x and $\hat{s}$. That is, whether the server yields (e,h) using her encrypted coefficients on $y \leftarrow x$ and randomness $\hat{r}$. If so, she puts x in the intersection set.

Figure2. Steps for Protocol PM-Malicious-Server

V. IMPLEMENTATION

The scenario we use for this protocol is client and server based. We decided to implement Pinkas Algorithm for malicious server by using an Android application which we built using Eclipse IDE. The name of the android application is Online Dating. Our application basically tells if two people have any common interest without disclosing their private information to each other. Only the client learns about the intersection set and server learns nothing. Since we have implemented an online dating application we used the logic that each number represents a unique interest. So instead of entering the interests a user enters the number which will represent an interest. Let us say that Music=1, Sports=2, Dancing 3 and so on. These numbers will be inputs of server and client. And then the client encrypts these numbers using the encryption technique described below.

A. Mixed Multiplicative Homomorphic Encryption (MMH)

As we mentioned in the introduction we use Mixed Multiplicative homomorphic encryption scheme which uses p and q that are large prime no and m= p*q, here p and q are kept secret. The set of original plaintext messages is Zp= {x|x <= p}, Zm = {x|x <m} is the set of cipher text messages and Qp = {a|a ∉ Zp} be the set of encryption clues. For the encryption scheme we perform is on a plaintext say x which is from the set Z_p and let 'a' be any random no from Q_p then x=a *mod* p and then cipertext is calculated by y=Enc(x) = a *mod* m. The decryption is also performed in the same way as x= Dec(y) = y *mod* p.
Let us see an example: Let p=17 and q=13 then m=p*q=17*13= 221. Let x_1=8, then Enc (8) = 59 and let x_2= 2 then Enc (2) = 36. So (59*36) *mod* 221= 135
So when we decrypt 135 we get 16= 135 *mod* 17 which is same as the multiplication of x_1*x_2= 8*2= 16

The advantages of using this scheme is that it's very fast and doesn't take time for long values of p and q. Also this type of encryption can be performed in real time as encryption function is called only once

The hash functions that we use for equation (2) and for randomization are SHA1 and MD5 as both of them give good performance results and also they are quiet secure. For the bit size of p and q we vary our results

B. Platform details

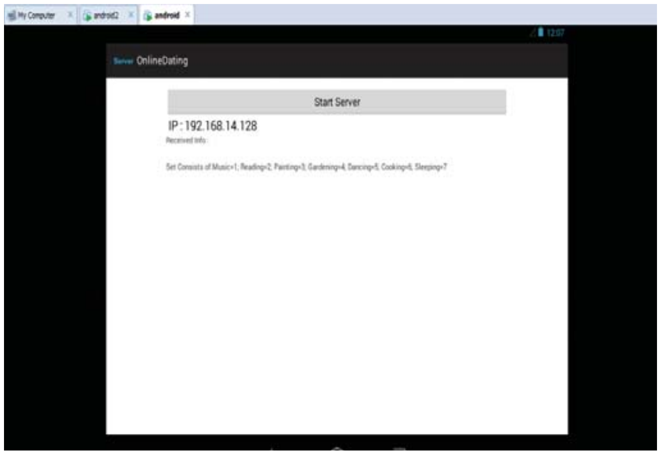
We developed the protocol as client server TCP/IP model on Windows 7 operating system. The table below gives a brief of implementation details

Table I
Platform Details

Programming Language	**Java**
Network Model	TCP/IP
Operating System	Windows 7 64bit
RAM	8GB
CPU	Intel i5 2.50GHz
Workstation	VMware 10.0.1
Android Version	X86-4.3.iso
Platform	Eclipse ADT Build-v22.3.0

. The following steps will show step by step what happens when we run our application

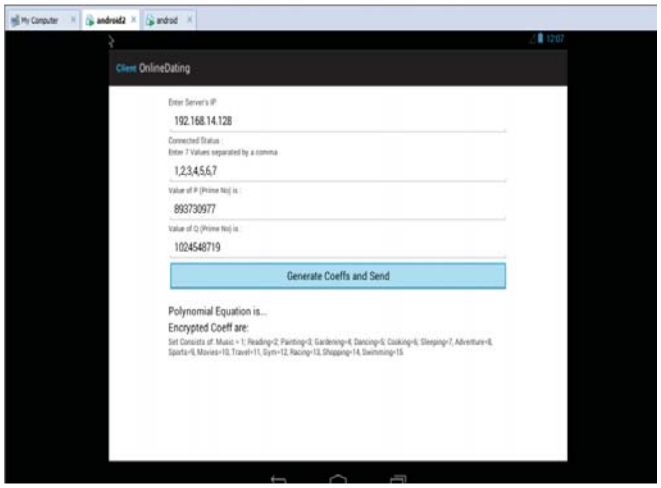
- We launch our application in eclipse IDE using VMware workstation on which two android iso are already installed
- We run the application, one of them is the server and other one is the client.
- The figure below shows the server and also displays the set of the server.



IP : 192.168.14.128
Received Info :
Set Consists of Music=1; Reading=2; Painting=3; Gardening=4; Dancing=5; Cooking=6; Sleeping=7

Figure 3.Server Interface

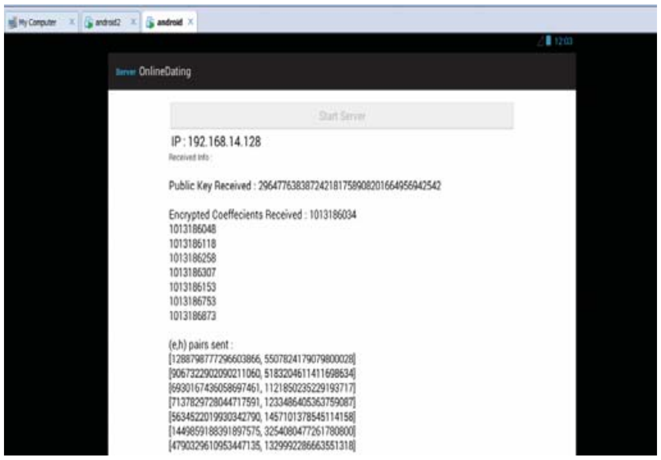
- Enter the IP address on the client side to connect it to the client. (IP address of server is displayed on the server side). In this case it is 192.168.14.128



Enter Server's IP
192.168.14.128
Connected Status :
Enter 7 Values separated by a comma
1,2,3,4,5,6,7
Value of P (Prime No) is :
893730977
Value of Q (Prime No) is :
1024548719
Polynomial Equation is...
Encrypted Coeff are:
Set Consists of: Music = 1; Reading=2; Painting=3; Gardening=4; Dancing=5; Cooking=6; Sleeping=7, Adventure=8, Sports=9, Movies=10, Travel=11, Gym=12, Racing=13, Shopping=14, Swimming=15

Figure 4.Connecting to Server

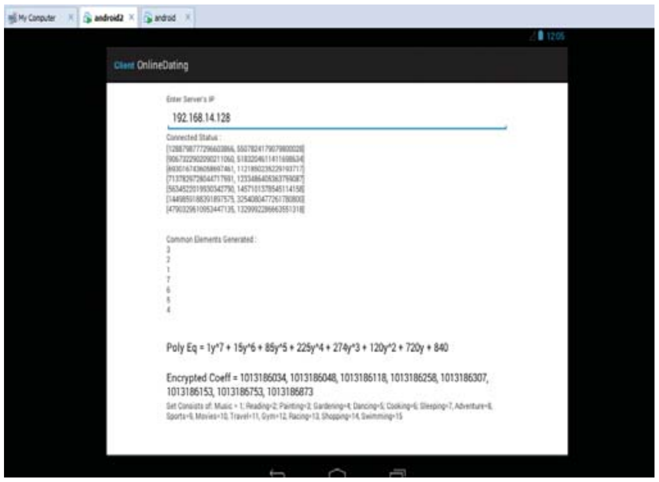
- As we can the client enter the IP address od server and then enters 7 numbers each of which represents his interests. The values of p and q are automatically generated. Then we hit the button "generate coeff and send"
- Once the connection is established the server and the client start communicating. In other words the polynomial equation is generated as mentioned in equation (1) and coefficients of the equation are being encrypted using mixed multiplicative homomorphic encryption and sent to the server
- All the information that is being generated on the server side can be seen on the screen. It receives the public key, encrypted coefficients and it generates the (e.h) pair as described in the equation (2)



```
IP : 192.168.14.128
Received Info :

Public Key Received : 296477638387242181758908201664956942542

Encrypted Coeffecients Received : 1013186034
1013186048
1013186118
1013186258
1013186307
1013186153
1013186753
1013186873

(e,h) pairs sent :
[1288798777296603866, 5507824179079800028]
[9067322902090211060, 5183204611411698634]
[6930167436058697461, 1121850235229193717]
[7137829728044717591, 1233486405363759087]
[5634522019930342790, 1457101378545114158]
[1449859188391897575, 3254080477261780800]
[4790329610953447135, 1329992286663551318]
```

Figure 5.Output on server side

- The (e,h) pair are then sent to the client and it decrypts the pair and find the common intersection
- In the client output we can see the polynomial equation, the encrypted coefficients, the (e,h) pair received and finally the common intersection set

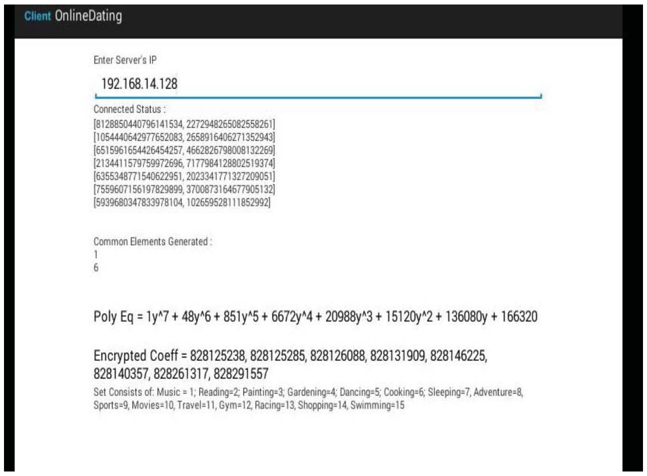


```
Enter Server's IP
192.168.14.128
Connected Status :
[1288798777296603866, 5507824179079800028]
[9067322902090211060, 5183204611411698634]
[6930167436058697461, 1121890235229193717]
[7137829729804471769, 1233486405363799087]
[5634522019930342790, 1457101378545114158]
[1449855188391897575, 3254080477261780800]
[4790329610953447135, 1329992286663551318]

Common Elements Generated :
3
2
1
7
6
5
4

Poly Eq = 1y^7 + 15y^6 + 85y^5 + 225y^4 + 274y^3 + 120y^2 + 720y + 840

Encrypted Coeff = 1013186034, 1013186048, 1013186118, 1013186258, 1013186307,
1013186153, 1013186753, 1013186873
Set Consists of: Music = 1; Reading=2; Painting=3; Gardening=4; Dancing=5; Cooking=6; Sleeping=7, Adventure=8,
Sports=9, Movies=10, Travel=11, Gym=12, Racing=13, Shopping=14, Swimming=15
```

Figure 6. Output on client side

- Here we see that on client side we get the intersection set as (1, 2, 3, 4, 5, 6, 7). Which means the common interest of these two people were music, painting, reading, gardening, dancing, cooking and sleeping.. We have assigned a number to each of the interest that a person has, this way it is easy for us to encrypt the number rather than encrypting the whole word.

Let us take another example where we begin our calculations by first selecting two sets, let the client have the set C→ (1,12,13,14,6,9,11) and server be composed of S→ (1,2,3,4,5,6,7). The client has to select a public key and two large prime that are generated automatically. Now the client constructs polynomial equation using equation (1) and we get the coefficients and then these coefficients are encrypted using mixed multiplicative homomorphic encryption. After encrypting these coefficients we send them over to the server and using homomorphic encryption it computes the set (e, h) using equation (2) and sends it to the client. Now the client decrypts e to find out r and then matches with its own set, if any element is common then the client puts it in the intersection set. Here we get the common intersection as (1,6).



```
Enter Server's IP
192.168.14.128
Connected Status :
[8128850440796141534, 2272948265082558261]
[1054440642977652083, 2658916406271352943]
[6515961654426454257, 4662826798008132269]
[2134411579759972696, 7177984128802519374]
[6355348771540622951, 2023341771327209051]
[7559607156197829899, 3700873164677905132]
[5939680347833978104, 1026595281118529992]

Common Elements Generated :
1
6

Poly Eq = 1y^7 + 48y^6 + 851y^5 + 6672y^4 + 20988y^3 + 15120y^2 + 136080y + 166320

Encrypted Coeff = 828125238, 828125285, 828126088, 828131909, 828146225,
828140357, 828261317, 828291557
Set Consists of: Music = 1; Reading=2; Painting=3; Gardening=4; Dancing=5; Cooking=6; Sleeping=7, Adventure=8,
Sports=9, Movies=10, Travel=11, Gym=12, Racing=13, Shopping=14, Swimming=15
```

Figure7. Client Output

VI. PERFORMANCE

The above results are when the size of p and q is 32 bits. We ran more tests to observer the run time for 16, 32, 128, 256, 512 and 1024 bits. We repeated each test several number of times to get an average result. We observed that as the no bits keep on increasing the time also keeps on increasing which means it's directly proportional to time. But the increase in time is quiet small which won't result in many delays. Keep in mind that for all the results below we used SHA1 as the hash algorithm. The results were similar when we used MD5 algorithm. The graph below shows us the results.

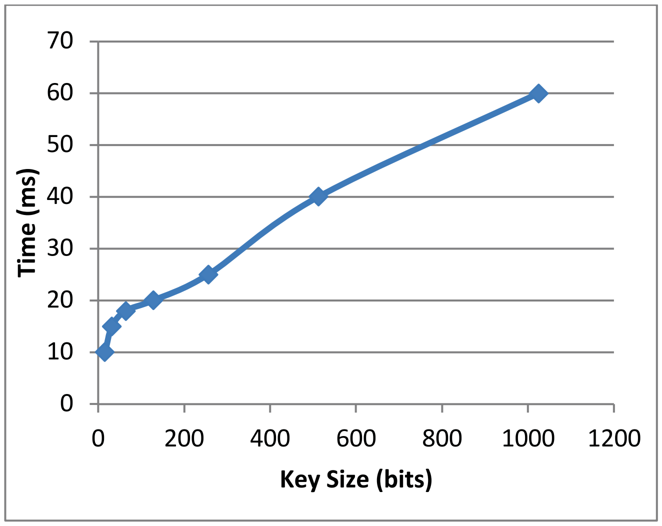


Figure 8. Time to generate encrypted coefficients

The above results are when we use SHA1 algorithm. But if we change the hash function to MD5 we see almost similar results as shown in graph below

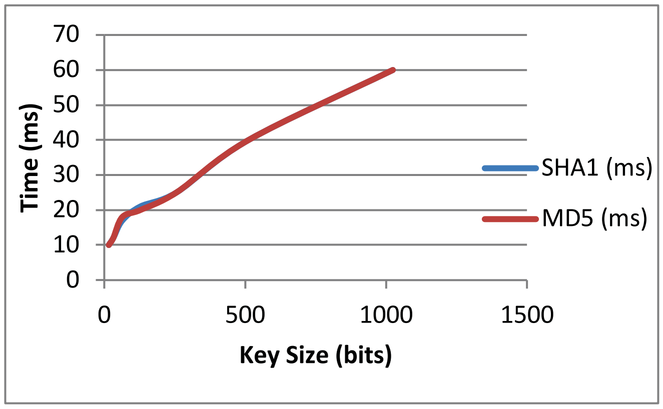
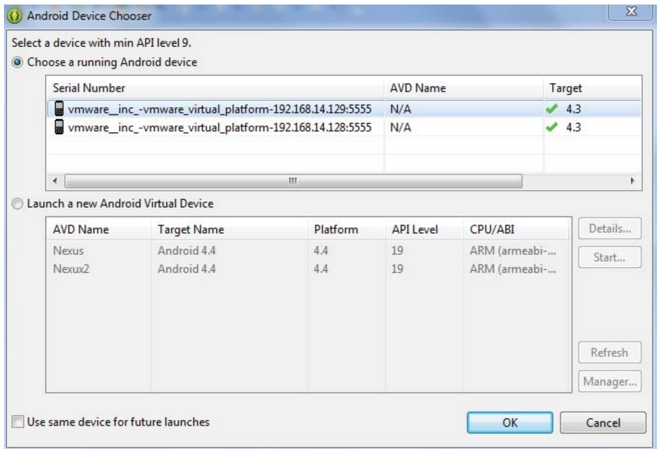


Figure9. SHA1 v/s MD5

While implementing this application we faced a major problem that was the emulator on Eclipse IDE used to take a long time to boot up which was a hindrance as we wanted quick results. To fix this problem instead of running our application on emulator we ran our application on VMware workstation. All we did was we installed android-x86-iso on VMware workstation and connected our project on Eclipse

IDE to Android-x86-iso installed on VMware. Once you install android-x86-iso on VMware workstation go to terminal emulator and run the command netcgf to know the IP address of the device. Then open the command prompt and go the android sdk platform-tools and run adb connect <ip address>. This will connect my android device on VMware to my eclipse. Go to eclipse and run the project run→android application→VMware. Select the machine where you want to run your application. The figure below shows what the screen will look like.



Android Device Chooser

Select a device with min API level 9.

(●) Choose a running Android device

Serial Number	AVD Name	Target
vmware__inc_-vmware_virtual_platform-192.168.14.129:5555	N/A	✓ 4.3
vmware__inc_-vmware_virtual_platform-192.168.14.128:5555	N/A	✓ 4.3

( ) Launch a new Android Virtual Device

AVD Name	Target Name	Platform	API Level	CPU/ABI
Nexus	Android 4.4	4.4	19	ARM (armeabi-...
Nexux2	Android 4.4	4.4	19	ARM (armeabi-...

☐ Use same device for future launches

Figure 10. Running Application on Eclipse

After this little experiment there was a huge difference in the run time of our android application. The table below will show our results.

Table II
Compares the result of run time when application was run on an emulator v/s application running on VMware Workstation

No of Repetition	**Application in VMware (sec)**	**Application in Eclipse Emulator(sec)**
1	55	240
2	49	310
3	64	200
4	50	380

VII.CONCLUSION

In this paper we used PSI method to find the common intersection of two sets without reveling actual sets. This scheme is useful as it keeps the sets of two parties secret, thus implementing privacy and integrity. To implement PSI we used PM-Malicious-Server protocol commonly known as Pinkas algorithm.

We made an android application to show our computation of this protocol, as smartphones are the new platform for privacy concerns. Instead of using any complex encryption technique we used MMH, which is one of the fastest and easiest encryption techniques. But MMH encryption method is susceptible to plaintext attack.

This type of privacy scheme is important in today's world as smartphones are becoming popular and people use it to share their sensitive information. However, there is a long way to go before we reach optimum results.

REFRENCES

[1] Emiliano De Cristofaro and Gene Tsudik: "*Practical Private Set Intersection Protocols with Linear Computational and Bandwidth Complexity* ". University of California, Irvine.

[2] Yaping Li, J. D. Tygar and Joseph M. Hellerstein: "*Private Matching*", Intel Research Berkeley.

[3] G. Sathya Narayanan, T. Aishwarya, Anugrah Agrawal, Arpita Patra, Ashish Choudhary, and C. Pandu Rangan: "*Multi Party Distributed Private Matching, Set Disjointness and Cardinality of Set Intersection with Information Theoretic Security*".

[4] Moni Naor and Benny Pinkas: "*Oblivious transfer and polynomial evaluation*". InProc. 31st Annual ACM Symposium on Theory of Computing, pages 245{254, Atlanta, Georgia, May 1999.

[5] Ronald Fagin, Moni Naor, and Peter Winkler: "*Comparing information without leaking it*". Communications of the ACM, 39(5):77{85, 1996.

[6] Oded Goldreich: "*Secure multi-party computation*". In Available at Theory of Cryptography Library, http://philby.ucsb.edu/cryptolib/BOOKS, 1999.

[7] Yossi Azar, Andrei Z. Broder, Anna R. Karlin, and Eli Upfal: "*Balanced allocations*". SIAM Journal on Computing, 29(1):180{200, 1999.

[8]. Yehuda Lindell and Benny Pinkas: "*Secure Two-Party Computation via Cut-and-ChooseOblivious Transfer*". Theory of Cryptography Lecture Notes in Computer Science Volume 6597, 2011, pp 329-346.

[9] Helger Lipmaa and Veri_able :"*Homomorphic oblivious transfer and private equality test*". In Advances in Cryptology|ASIACRYPT 2003, pages 416{433, Taipei, Taiwan, November 2003.

[10] Bala Kalyanasundaram and Georg Schnitger: "*The probabilistic communication complexity of set intersection*". SIAM J. Discrete Mathematics, 5(4):545{557, 1992.

[11] Andrei Z. Broder and Michael Mitzenmacher: "*Using multiple hash functions to improve ip lookups*". In IEEE INFOCOM'01, pages 1454{1463, Anchorage, Alaska, April 2001.

[12] Alexandre Evmievski, Johannes Gehrke, and Ramakrishnan Srikant: "*Limiting privacy breaches in privacy preserving data mining*". In Proc. 22nd ACM Symposium on Principles of Database Systems (PODS 2003), pages 211{222, San Diego, CA,June 2003.

[13] Justin Brickell and Vitaly Shmatikov :"*Privacy-Preserving Graph Algorithms in the Semi-honest Model*". The University of Texas at Austin, Austin TX 78712, USA.

[14] Keita Emura, Atsuko Miyaji and Mohammad Shahriar Rahman "*Eficient Privacy-Preserving Data Mining in Malicious Model*".

[15] Michael J. Freedman, Kobbi Nissim and Benny Pinkas: "*Efficient Private Matching and Set Intersection*". Advances in Cryptology - EUROCRYPT 2004.

[16] Bo YANG, Aidong SUN, Wenzheng ZHANG: "*Secure Polynomial Computation in the Semi-honest Model*". Journal of Computational Information Systems 6:6(2010) 1907-1921.

[17] Craig Gentry:"*A Fully Homomorphic Encryption Scheme*". A Dissertation Submitted To The Department Of Computer Science And The Committee On Graduate Studies Of Stanford University September 2009

[18] Yan Huang, Peter Champan and David Evans "*Privacy-Preserving Applications on Smartphones*". In 6th USENIX Workshop on Hot Topics in Security, San Francisco, 9 August 2011.

[19] Seny Kamara,Payman Mohassel, Mariana Raykova, Saeed Sadeghian: "*Scaling Private Set Intersection to Billion-Element Sets*".

[20] David P. Woodruff: "*Near-Optimal Private Approximation Protocols via a Black Box Transformation*".

[21] Josep Domingo-Ferrer: "*A Provably Secure Additive and Multiplicative Privacy Homomorphism*", in A.H. Chan and V. Gligor (Eds.): ISC 2002, LNCS 2433, pp. 471–483, 2002

[22] K.Gopi:"*Homomorphic Encryption Schemes for Secure Packet Forwarding in Mobile Ad hoc Networks*".

[23] Aditya, Riza, Boyd, Colin, Dawson, Edward, Lee, Byoungcheon, & Peng: " Multiplicative Homomorphic E-Voting", in Canteaut, A & Viswanathan, K (Eds.) Progress in Cryptography.