Privacy-preserving contact comparison for social networks

Gokay Saldamli[1], Levent Ertaul[2] and Mayur Gala [2]
[1]San Jose State University, San Jose, CA, USA
[2]California State University, East Bay, Hayward, CA, USA
gokay.saldamli@sjsu.edu, levent.ertaul@csueastbay.edu, mayurgala@horizon.csueastbay.edu

***Abstract*— Privacy preserving content sharing has become one of the most desired feature of most social, mobile or internet applications. Today the amount of private data flowing in the internet or at rest is bigger than ever before and its size is growing with an unpresented pace. Most of the users as well as the regulators are concerned about privacy of that data. Most applications do not need raw data but instead require some data analytics. In reality, desired analytic functions are mostly simple ones such as accumulations, mean or average calculations, unions or intersections. In this study, we focus on private intersection of data owned by different parties without revealing the data that is not in the intersection. We propose an efficient method of privacy preserving sensitive information sharing that are usable and practical in the real world scenarios. Our method guaranties that communicating parties learn no information beyond what they are required to know. There are wide set of applications ranging from social networking to national security using privacy preserving data intersection.**

***Index Terms*—Privacy preserving computations, location, contact sharing.**

I. INTRODUCTION

The growing concern related privacy is increasing and many people are concerned about their private data as the private data is online and readily accessible ever before. Privacy can be defined as the ability of an individual/group to separate themselves or any information about themselves and to share the information selectively. So selective sharing is important. Sharing only a selected part of information and to selected set of people. So it is important to have a method to achieve this in an efficient manner. A common privacy preserving sharing example would be a user, Bob wants to find common contacts with another user, Alice. However, Alice doesn't want to share her all of the contacts but she is fine showing Bob the common contacts. The question of how can they check for common contact without revealing their other contacts so the solution to this technique is Privacy-Preserving Sharing of Sensitive Information **(**PPSSI). The applications of this simple story telling can be mind blowing. For instance, the following are some of the direct application:

- *Aviation Safety:* There are many secret list with the department of Homeland Security. The list like Terror Watch list (TWL) [2]. The Department of Homeland security depending on these list checks whether any passenger on flight from/to USA must be allowed or denied. So the information about a flight is given to DHS by the airlines. The information surrendered contains a lot of private information about the passengers. This includes a liability for the innocent passengers. So ideal case should be that the DHS should get only the information about the passenger which are on any of the watch list and the other information should not be disclosed. (For recent incidents about sensitive information stolen by government officials check [3]).
- *Law Enforcement:* For an investigating agency like FBI often require to get information about a suspect for various places like IRS, DMV, or suspect's employer. FBI can't disclose the subject of its investigation and all the other agencies can't trust the FBI to extract only relevant information and share the whole set of the data which includes other unsuspected people. PPSSI method would also be beneficial in this use case.
- *Health Care:* Hospitals have lot of private information regarding their patients but many other department and agencies would a particular information regarding a patient with disclosing who that patient is. In that case also the PPSSI would be useful.

There are many more example in which the PPSSI technique would be a perfect fit and would be able to solve many issues. Some other examples include Profile Matching algorithm, sharing interest from smart phone [1, 5, 6, 7].

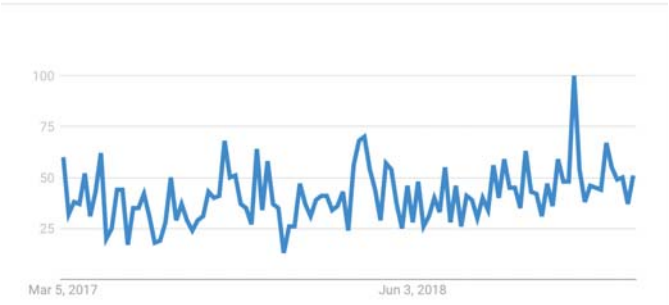

Fig. 1. Google trends for "privacy violation".

Fig. 1 shows the google trends graph on "Privacy violation" as a search term. Notice that the privacy awareness are in a

rising trend. More people are concerned about privacy more than ever before.

Private set intersection or privacy preserving sharing are methods of achieving privacy using cryptographic primitives. We take [1] as a baseline where the authors modeled in the form of a normal database querying application with a server and a client. Where server has the database and client performs query. The main building blocks of their protocol were the efficient private set intersection (PSI) techniques. We used one of the protocol defined in [1] and implemented a working version of that protocol into a mobile application to compare contacts targeting possible applications including social networks.

We organize the paper as follows: we briefly explain the base protocol in Section II. We go through the implementation details in Section III. After discussing some major security and performance analysis in Section IV we follow by conclusion in Section V.

II. PPSSI PROTOCOL

The protocol we refer to in this paper is the protocol designed by [1]. We use this protocol to implement as simple common contact android application where users can intersect their contact privately. Following are three main steps involved in the exchange of between client and server.

1. The Client and server do Oblivious Computation and exchange initial parameters.
2. Using that initial parameters the server will then calculate and create the Encrypted Database and send that to the client.
3. Client does the lookup procedure using the initial parameters.

Table 1:Notation

$ctr_{j,1}$	# of times where $val_{j',l} = val_{j,l}$ for all $j' \leq j$
$tag_{j,l}$	Tag for $attr_l$, $val_{j,l}$
$k'_{j,l}$	Key used to encrypt k_j
$k''_{j,l}$	Key used to encrypt index j
$ek_{j,l}$	Encryption of key k_j
$eind_{j,l}$	Encryption of index j
$attr_l$	l-th attribute in the database schema
R_j	j-th record in the database
$val_{j,l}$	Value in R_j corresponding to $attr_l$
k_j	Key used to encrypt R_j
er_j	Encryption of R_j
$tk_{j,l}$	Token evaluated over $attr_{j,l}$, $val_{j,l}$

The main use case for using our protocol is where there is a party, client, who wants to compare his contacts with other party called the server. They both want to compare the contacts but they don't want to compromise the information about their rest of the contact. Hence the client and server both know the structure of contacts database. They both also share a common prime number *p* and a generator *g*. In order to have a compact presentation we summarize the notation used in the protocol in Table 1.

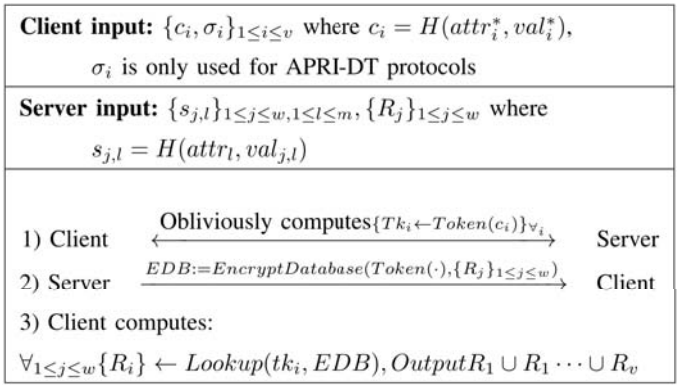

Fig. 2. Online of the PPSSIU approach [1}

The client and server communicate initially and exchange some initial parameter. Then after receiving the initial parameter the server generates the EncryptedDatabase called EDB and send that to the client. During that time the client generates tokens for all its (Attribute, Value) pairs and after receiving the EDB the client runs the lookup procedure on the EDB. The tokens which match are the common contacts so that the client can then decrypt only those contacts. Note that not all the encrypted database but only the entries which exist for both are revealed and the server would not know the contacts which were matched.

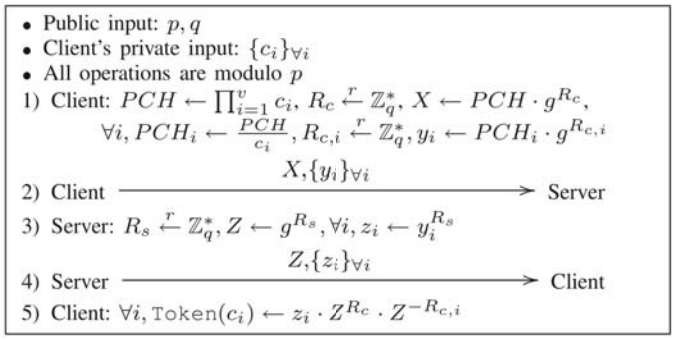

Fig. 3. Oblivious computation of Token(.) using DT10-1 [1,4]

Figs. 2 and 3 gives the details of the protocol [1] which we will be using for the implementation purpose. In step 1 of Fig. 2, the client and server do the oblivious computation of the Token. After step 1 the client has $tk_i = Token\ (c_i)$, where $c_i = H(attr^*_i, val^*_i)$. The server is not aware of tk_i. The token is calculated using a PSI-DT protocol [4]. In Step 2 the server runs EncryptDatabase(EDB) procedure that is described in Algorithm 1. The EDB is then sent to client in step3 then finally in step 4 client executes the lookup procedure which showed in Algorithm 2. Using the token over EDB the client retrieves the set of records.

Oblivious Computation shown in appendix of [5] the authors had selected PSI-DT protocol from [4] denoted as DT10-1 shown in Fig. 3. The token calculation can be summarized in short with Eqn. (1).

EncryptDatabase procedure is illustrated in Fig. 4. In this after the oblivious computation the server encrypts the database using Algorithm 1. In this the server shuffles all the records and then encrypt each record then for each attribute value pair in the record, it calculates the token and using that token it encrypts the key and index number of the row encrypted and generate a LTable entry with the hash of the token, encrypted key of row, and encrypted index. All the encrypted row and Ltable entries makeup the Encrypted database.

Algorithm 1: `EncryptDatabase` Procedure.

input : Function `Token`$(\cdot)$ and record set $\{R_j\}_{1\le j\le w}$
output: Encrypted Database **EDB**

Shuffle $\{R_j\}_{1\le j\le w}$
$maxlen \leftarrow$ max length among all R_j
for $1 \le j \le w$ **do**
  Pad R_j to $maxlen$;
  $k_j \xleftarrow{r} \{0,1\}^{128}$;
  $er_j \leftarrow Enc_{k_j}(R_j)$;
  for $1 \le l \le m$ **do**
    $hs_{j,l} \leftarrow H(attr_l, val_{j,l})$;
    $tk_{j,l} \leftarrow$ `Token`$(hs_{j,l})$;
    $tag_{j,l} \leftarrow H_1(tk_{j,l}||ctr_{j,l})$;
    $k'_{j,l} \leftarrow H_2(tk_{j,l}||ctr_{j,l})$;
    $k''_{j,l} \leftarrow H_3(tk_{j,l}||ctr_{j,l})$;
    $ek_{j,l} \leftarrow Enc_{k'_{j,l}}(k_j)$;
    $eind_{j,l} \leftarrow Enc_{k''_{j,l}}(j)$;
    $\mathbf{LTable}_{j,l} \leftarrow (tag_{j,l}, ek_{j,l}, eind_{j,l})$;
  end for
end for
Shuffle **LTable** with respect to j and l;
$\mathbf{EDB} \leftarrow \{\mathbf{LTable}, \{er_j\}_{1\le j\le w}\}$;

After the EDB is generated it is sent to the client, it goes through a lookup procedure on the EDB. The Lookup procedure is given in Algorithm 2. The lookup procedure is straight forward where the client generates its set of token with the Attribute value pair that client has. The token generation is done after the oblivious computation step is finished. The client loops through all the tags which received from the server in the EDB. If the token matches it mean that it is an intersection. The client decrypts the value of index and key and then using that index it extracts the specific record and decrypt it using the key extracted from the LTable.

Algorithm 2: `Lookup` Procedure.

input : Search token tk and encrypted database $\mathbf{EDB} = \{\mathbf{LTable}, \{er_j\}_{1\le j\le w}\}$
output: Matching record set $\mathbf{R}$

$ctr \leftarrow 1$;
while $\exists tag_{j,l} \in \mathbf{LTable}\ s.t.\ tag_{j,l} = H_1(tk||ctr)$ **do**
  $k'' \leftarrow H_3(tk||ctr)$;
  $j' \leftarrow Dec_{k''}(eind_{j,l})$;
  $k' \leftarrow H_2(tk||ctr)$;
  $k \leftarrow Dec_{k'}(ek_{j,l})$;
  $R_j \leftarrow Dec_k(er_{j'})$;
  $\mathbf{R} \leftarrow \mathbf{R} \cup R_j$;
  $ctr \leftarrow ctr + 1$;
end while

$$Token(c) := \left(\frac{(\Pi_{i=1}^{v} c_i) \cdot g^{R_c}}{c}\right)^{R_s} \ mod\ p \qquad (1)$$

III. Implementation

We have implemented the PPSSI method mentioned in [1] as an android application to compare contacts. We use AES (with key sizes 128, 192 and 256) encryption [11] and SHA1 hashing [12]. During the key generation for AES in android there is a issue with android in SecureRandom function. It keeps changing whenever the application is restarted so the Secure random function used is with a seed string. The SecureRandom function works well in Desktop version of Java but when using the SecureRandom Function while decryption it failed to generate the same key. It gives the Bad Padding Exception [10].

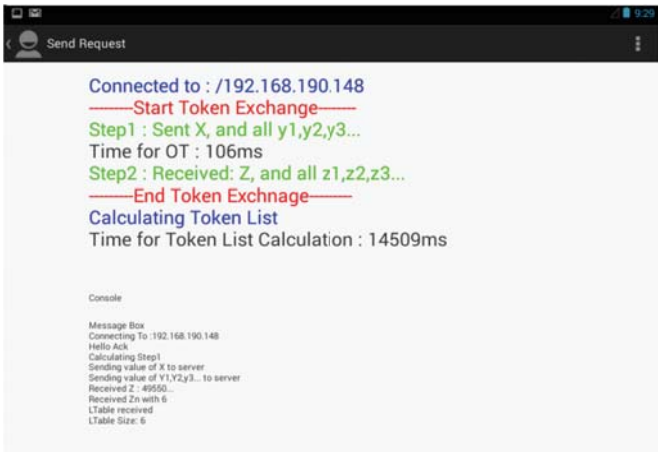

Fig. 4. Reply from a friend's request

The public input are hardcoded into the application. Our main objective was to check the performance of the protocol on a mobile platform. For development, the Android KitKat was used and most of the functions we current and not deprecated. The minimum API level for the Network Communication was APILEVEL 9. Eclipse with the whole ADT bundle was used for the development at the beginning and the during the end of development switched to Androids IDE called the Android Studio [9][10]. For testing device VMware was used. We Installed Android_x86 project ISO in a VMware and connected to the VMware using the "adb" connect command. The ram allocated to the VM was 1GB and single processor.

The first thing we developed was client server communication on the two mobile devices. So the requirement for this app is that both the client and server should be in a same network and their IP should be routable.

When the user open the application, it shows two options namely: send request to friend or receive request form friend. If the "send req to friend" option is selected then the user will be presented by the input box to enter IP address of the other user and there will be a button to connect. The "Send req to Friend" option is displayed. If the user selects "Recv request from friend" then he will be displayed with his own IP address so that the other friend can connect using this displayed IP address. The "rec req from friend" option is displayed in Fig. 4. In this way

the two user are connected. So the one who is sending the request in the client in terms of protocol and the other user who is receiving the request is the server. Here on for ease of explanation the two user will be referred as client and server for the rest of this paper.

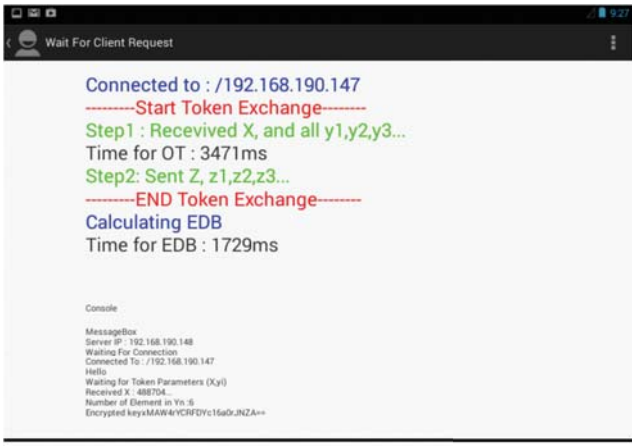

Fig. 5. server response

After the connection is established the client and server carry out the oblivious computation. For the initial calculation BigInteger [13] is used as it integer and Long datatypes were small. Then as the end of oblivious computation the BigInteger is converted to long to save memory. During the first testing of the app the contacts were hardcoded and both the client and server had 6 contacts each with 3 common contacts. For understanding and easy debugging purpose there is a message box (TextView [14] in Android terms) called console. In Console the all the steps taken and progress of protocol is displayed (see Fig. 5).

IV. Security and Performance Analysis

The authors in [1] do not mention any way of transferring the EDB to the client. In Algorithm 1 where the server calculates the EDB they encrypt the key and index using the hash of token (tag) as key, and tag is one of the entry in the LTable. This implies that they are providing the key with the encrypted values. Hence the client can access all the records.

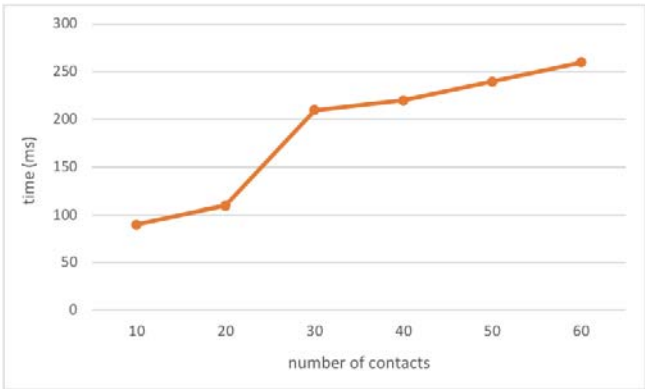

Fig. 6. Client oblivious transfer

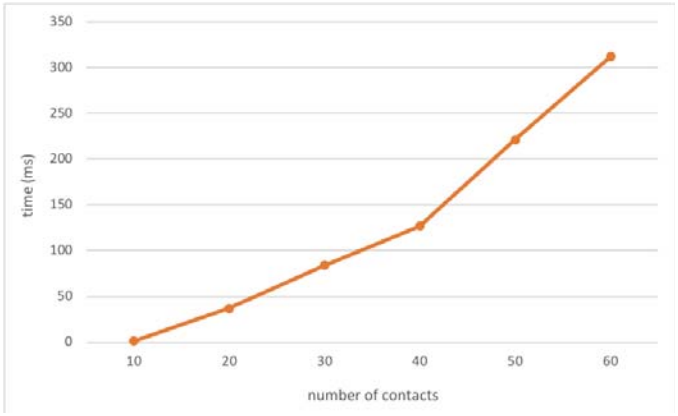

Fig. 7. Server oblivious transfer

The protocol execution time is high for larger dataset. For around 50 contacts the wait time is around 5 minutes which could be frustrating for users and not scalable and for large scale systems as seen in Figs. 6 and 7. The major time consuming part is oblivious transfer.

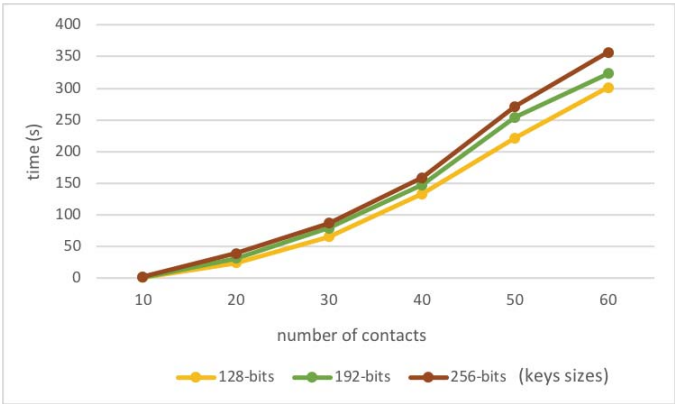

Fig. 8. Client lookup performance

For evaluating the performance of the algorithm, first thing we checked is how many contacts it can handle (dataset size). So we varied the size of contact list to check the performance with respect to the Oblivious Transfer and lookup or EDB generating procedure for client and server respectively. There was huge difference when the size of dataset increased. But the any point the memory requirement did not increase significantly as all the calculations were atomic at a time only one entry was processed as seen in Figs. 8. More efficient way would be to do the calculation in a multithreaded environment that will increase a performance but that is a future scope as many devices still are single core they might not be able to utilize the full potential of multithreading. But there are increasing number of devices coming out with multi core CPU. So it would be lot faster to implement a multithreaded version of this protocol.

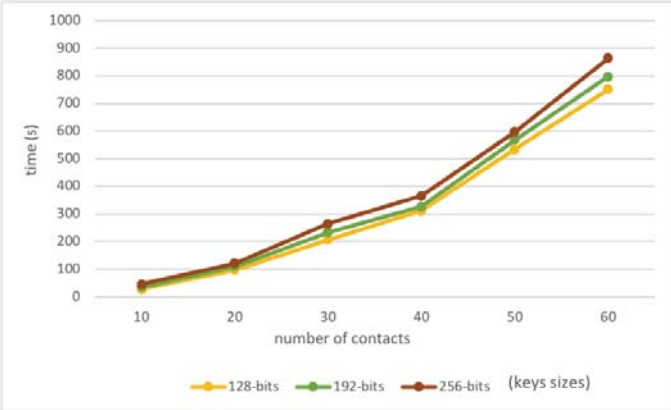

Fig. 9. Total execution time for client

Figs. 10 and 11 shows that the change in size of AES encryption key did not impact the time of execution but we noticed that the protocol is not good for scaling because as we increase the number of contacts the time taken to execution also increased. This is also evident looking at other figures also. All the figure had the similar pattern. In Fig. 6 there was an anomaly in the time, but the reason for that would be the CPU load if the CPU cycles are free then it the execution time is less.

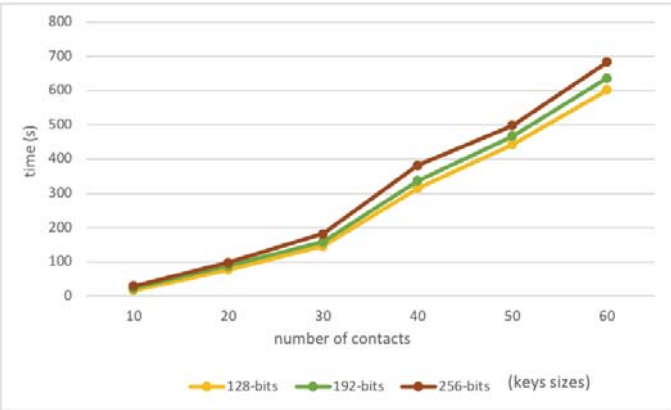

Fig. 10. Total execution time for server

As there was no way to keep the execution environment perfect always there were some time variation seen so they have been averaged out. In theory the Key size for AES encryption should not change the execution time of algorithm. One more thing observed was the time taken by Server to calculate the EDB and the time taken by the Client to do the lookup was always almost the same.

V. Conclusion

In this paper we studied a PPSSI protocol and implemented it on Android Platform. It was interesting to study the protocol and implement the same in an android app because of which we got to learn new things about security like the Private set intersection approach and PPSSI and learnt how important it is to implement the protocol as specified else it may generate more security concerns. This approach has some small flaws in terms of Security and performance. There is some future scope in improving the implementation by using multi-threaded approach. There is also need to improve security as the key is given in LTable there should be a different approach used to share the key.

REFERENCES

[1] E. De Cristofaro, Y. Lu, G. Tsudik. "Efficient Techniques for Privacy-Preserving Sharing of Sensitive Information. " In Proceedings of Trust and Trustworthy Computing (Trust 2011), LNCS 6740, Springer, pp. 239-253, 2011.

[2] J. P. Bjelopera, B. Elias and A. Siskin, "The terrorist screening database and preventing terrorist travel," [Online] Available: https://fas.org/sgp/crs/terror/R44678.pdf, November 2016..

[3] Caslon Analytics: Consumer Data Losses. http://www.caslon.com.au/datalossnote.htm.

[4] E. De Cristofaro and G. Tsudik. "Practical private set intersection protocols with linear complexity." In Proceedings of the 14th international conference on Financial Cryptography and Data Security (FC'10), Springer, pp. 143-159, 2010.

[5] Shishir Nagaraja, Prateek Mittal, Chi-Yao Hong, Matthew Caesar, and Nikita Borisov. "BotGrep: finding P2P bots with structured graph analysis." In Proceedings of the 19th USENIX conference on Security (USENIX Security'10). USENIX Association, Berkeley, pp. 7-7, 2010.

[6] Emiliano De Cristofaro, Anthony Durussel, and Imad Aad. "Reclaiming privacy for smartphone applications." In Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications (PERCOM '11). IEEE Computer Society, pp. 84-92, 2011.

[7] Elie Bursztein, Mike Hamburg, Jocelyn Lagarenne and Dan Boneh. "OpenConflict: Preventing Real Time Map Hacks in Online Games." In proceedings of the 2011 IEEE Oakland Security and Privacy conference, pp. 506-522, 2011.

[8] Android Developers. [Online] Available: http://developer.android.com/guide/index.html.

[9] Android Developer Tools. [Online] Available: http://developer.android.com/tools/index.html

[10] Bad Padding Exception fix. [Online] Available: http://blog.kchandrahasa.com/blog/2013/08/09/android-4-dot-2-and-javax-dot-crypto-dot-badpaddingexception-pad-block-corrupted/

[11] Joan Daemen and Vincent Rijmen. The Design of Rijndael. Springer-Verlag, Berlin, Heidelberg, 2002.

[12] Secure Hash Standard, FIPS 180-4. [Online] Available: https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf

[13] Oracle Java BigInteger, [Online] Available: http://docs.oracle.com/javase/6/docs/api/java/math/BigInteger.html

[14] Android TextView, [Online] Available: http://developer.android.com/reference/android/widget/TextView.html

[15] Secure Random Class java. [Online] Available: http://docs.oracle.com/javase/7/docs/api/java/security/SecureRandom.html

[16] BigInteger Android Developer, [Online] Available: http://developer.android.com/reference/java/math/BigInteger.html

[17] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. "Private information retrieval." J. ACM vol. 45, Issue 6, pp. 965-981, November 1998.

[18] Rakesh Agrawal, Alexandre Evfimievski, and Ramakrishnan Srikant. "Information sharing across private databases." In Proceedings of the 2003 ACM SIGMOD international conference on Management of data (SIGMOD '03), ACM, New York, NY, pp. 86-97, 2003.

[19] M. J. Freedman, K. Nissim and B. Pinkas. "Efficient Private Matching and Set Intersection," Advances in Cryptology - EUROCRYPT 2004, Lecture Notes in Computer Science, vol. 3027, Springer, pp. 1-19, 2004.