Private Contact Discovery

Mario Cornejo

May 2016

Abstract

Two users want to know which phone numbers they have in common, but they are very conscious that sharing their contact lists is a very bad idea. The users actually want to compute the intersection of private sets while neither should learn anything more that the intersection itself. This white paper presents a novel technique in order to achieve Private Set Intersection (PSI) based on Symmetric Searchable Encryption (SSE) taking into consideration a mobile setup.

1 Introduction

Nowadays, tech companies are deploying privacy-preserving protocols in their final users' products with a twofold intention: On the one hand, the user can enjoy its private communications and data privacy. On the other hand, the company cannot comply with courts' warrants to reveal data of its users because it is encrypted. An example of a user-oriented privacy-preserving protocol is Private Set Intersection, which can be applied in many situations and scenarios like the Tax authority wanting to learn whether any suspected tax evaders have bank accounts in some foreign bank. Clearly, none of both wants to reveal their list (neither the evaders nor the clients). The same principle applies for different-countries' intelligence agencies: they want to collaborate without revealing anything. In a family environment, sharing contacts could also be a useful tool to know if they have their family and friends' phone numbers up-to-date.

Private Contact Discovery A typical use case of PSI is a contact discovery protocol. This is the case of applications with social interactions. After creating an account, the user needs to send its contact list to the server, so that the server can identify which users are already enrolled to linked them. When the contact discovery is Friend-to-Friend, then it is a little trickier since a malicious user can manipulate her list of friends by populating it with a large amount of phone numbers in order to maximize the amount of information learned.

We based our solution keeping in mind the mobile setting. That is, smartphones with limited computational resources (old and new CPUs), network usage (for slow connections or download data quota) and trying to limit the user interaction (eg. does not have a keyboard or knowledge about cryptography).

The idea behind this protocol is based on [CT09, CT10]. Assuming that neither player aborts the protocol prematurely and both follow it correctly, it is possible to construct a mutual PSI by two instantiations of the one-way PSI. In [CJKT09] the authors introduce the idea of constructing a PSI from Identity-based Encryption (IBE) [BF01] and IBE-based Public-Key Encryption with Keyword Search [BDCOP04] (Searchable Encryption).

Fortunately[1], the research in Searchable Encryption was quite active since 2004, and it moved from public-key to Searchable Symmetric Encryption (SSE) which is more suitable in this case. A good comparison between these protocols can be found in [CGKO06]. To the best of our knowledge, the idea of using SSE as an underlying protocol for PSI is unexplored but mentioned in [KMRS14].

SSE schemes allow a client to encrypt a database, which in our case will be a contact list, in a way that enables an efficient search by keywords. SSE provides a good tradeoff between security, efficiency, and the ability to securely update the data after it has been encrypted and uploaded. After the encrypted database has been created, the owner can deliver tokens (or an authorization) to allow another user to query the database.

2 Protocol

The protocol herein is based on [CJJ+14]. A very efficient dynamic implementation of single-keyword search (SKS) based on a dictionary data structure is proposed for very large databases. On top of that, we make the queries verifiable to ensure that the server always answers correctly using [BFP16] technique which is basically using strong accumulators [CH12] to create a proof that an element is in the set or not.

Intuition This protocol is run by (at least) two parties Alice and Bob and one server Claude. The main idea is to run a simplified symmetric searchable encryption twice: one for each user. First Alice creates the encrypted database of her phone numbers and sends it to Claude who stores it. The database is a hash table containing an identifier for every phone number.

Later, Bob requests a secret key from Alice in order to create one identifier for each phone number that he has in his contact list. Then Bob sends his identifiers to Claude who computes the search using the encrypted database

[1]Fortunately, because no practitioner in its right mind would implement IBE on a smartphone, yet.

of Alice and the identifiers of Bob, and returns as output of that algorithm the intersection of the datasets to Bob. Claude does not learn anything of the phone numbers since the identifiers are created using a pseudorandom function PRF. The same protocol is run by Alice (possibly simultaneously) and at the end of the protocol both users have the common contacts.

In Figure 1, the intuition of the protocol is represented, skipping the verification proofs and details of the algorithms.

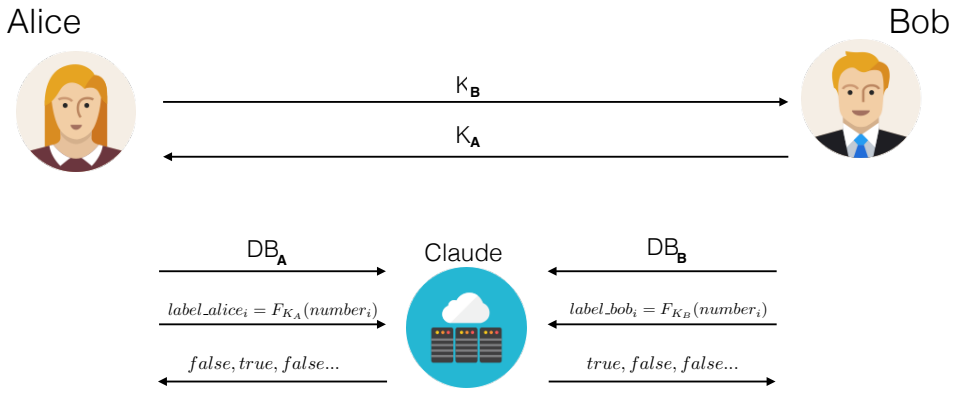

Figure 1: Simplified protocol

Private Contact Discovery with a the Semi-Honest Server The construction proposed here partially follows Cash's *et al.* [CJJ+14] work and a similar notation is used. It consists of a pair of algorithms: `Setup`, which the owner of the contacts list runs, and the protocol `Search`, which a third-party server runs jointly with another user.

The `Setup` takes as input a database (a contacts list) DB, and outputs the encrypted contact list `EDB` along with a secret key K that allows the user to compute a pseudorandom value for each entry of the contact list. The procedure is presented in algorithm 1 where F is a PRF. The functions `Add` and `Get` are part of a standard hash table data structure.

The `Search` algorithm is run between the server and a client. The client takes as input the secret key K (generated by the `Setup`) and a phone number $w \in \{0,1\}^*$ (normally a list of phone numbers). The client generates one identifier for each phone number that the client has and sends the identifiers to the server. The server takes as input `EDB` and the identifiers generated by the client. The server outputs the identifier if it was found in the database.

The protocol works as follows: Alice and Bob want to share their contact list. Let us assume that Alice will start the protocol. Alice executes the `Setup` algorithm with her contact list as input. Once finished, the key K

is send privately to Bob and the database is uploaded to the third-party server. Now, Bob uses the key K and the same derivation function as Alice to derive the identifier for each phone number that he has and sends them to the server. Upon reception, the server does a lookup (`Get`) on the encrypted database and, if the identifier is found, returns the identifier.

Algorithm 1 Setup algorithm

procedure SETUP(DB) ▷ DB owner
  $K \leftarrow \{0,1\}^{\lambda}$
  Allocate hash table H
  for each $w \in W$ **do**
    $\ell \leftarrow F(K, w)$
    `Add`(H, ℓ)
  end for
  Output (K, H)
end procedure

Algorithm 2 Search Algorithm: Client & Server side

procedure SEARCH(K, w) ▷ Client side
  $l \leftarrow F(K, w)$
  Send l to the server.
end procedure

procedure SEARCH(DB, l) ▷ Server side
  $d \leftarrow$ `Get`(DB, l)
  Output d
end procedure

Private Contact Discovery with multiple users The case of multiple users is similar to the case of two users with the difference that the server receives the identifiers of all the participants and the encrypted databases of at least two of the participants. He then computes the intersection for all the users and returns the values that match for every user.

Private Contact Discovery with Malicious Server In the case of the server acting maliciously and not following the protocol, he could deliver wrong messages. This is easily fixable using a Verifiable Hash Table. The main idea is to provide a proof that, when querying a key in the hash table, the returned element is the right one, and if the key is not present in the hash table, there is no associated element. The server's overhead complexity is $O(\log n)$ with respect to the size of the database and $O(n)$ for the user since he has to verify all the elements. An implementation of a Verifiable Hash

Table is presented in [BFP16]. Using this technique, the server answers the values along with a proof that the element is contained (or not) in the set. The server is forced to follow the protocol, otherwise it will be discovered by the participants at the moment of verification of the answers.

3 Security

The security of the protocol relies on the PRF and the encryption scheme. Intuitively[2], the Setup algorithm outputs the encrypted phone number and a pseudorandom value generated from the phone number. In any case, an attacker who gains information about the phone numbers is breaking the security of the PRF. In the Search algorithm, the client cannot learn anything from the encrypted database because K was generated before the creation of the database. Finally the server cannot learn any information about the phone numbers of the user because of the same argument given before: the server has to break the security of the PRF.

The information the users learn by the protocol is only the list contacts they have in common. The size of the database (and/or the number of records) is leaked to the server, as well as the number of phone numbers that the participants have in common.

Implementation & Assumptions For the implementation, HMAC-SHA1 primitive is used as PRF. HMAC-SHA256 or AES could have also been used, depending on the level of security and hardware capabilities.

The participants are considered honest-but-curious meaning that they follow the prescribed protocol but they want to learn as much as possible from it. Also, it is assumed that the participants are using their real contact list and not a fake one with more numbers in order to learn more numbers. This is an actual issue in practice. However, might be easily mitigated by adding a throttle policy in the server (if it is controlled by a trusted party) and thereby detecting users that are performing online attacks.

The size of the contact list used in the implementation is 5000 based on the statement in [Mar14]: *The average Android user has approximately 5000 contacts synced to their device.* Taking into consideration ITU's recommendation E.164 that limits the size of the phone numbers to 15, in the implementation, each phone number is modeled as 15 digits integer.

For this prototype, the implementation was static, nevertheless it is possible to extend this to a dynamic database, in which the user uploads only one time his encrypted contacts and he has the possibility of updating its contact list. Additionally, he can create and revoke keys on demand, managing parties allowed to calculate an intersection with a lot of flexibility. This is possible using a SSE multi-client scheme or similar methods.

[2] it is not intended to be a formal security proof

The prototype was written in Scala using Activator (SBT) as a build tool. Since the protocol is so simple, this could have been coded in any language. For a real implementation a native code is preferred (C-style) to profit of the hardware optimizations. The structure of the code is simple: There is a class `User` and a class `Server` which are instantiated with different settings in the `Test` class.

To run a small demo of the code, type:

```
$ cd PriCoD
$ activator run  #or 'sbt run' to run the main class
```

To run different kind of tests, type:

```
$ cd PriCoD
$ activator test  #or 'sbt test'
```

4 Discussion

Malicious users In a setting where the users are malicious but the server is semi-honest, to avoid an abuse by the users, it is possible to set a limit of queries that a user can do based on global statistics of the users. For example, if the average number of contacts in the system is 5000, then each user has the possibility of querying the intersections with databases for up to $5000 + \delta$ of his contacts each day. Using an algorithm that sets the value of δ for each user, it is possible to reduce the number of queries daily (or any time frame).

Computation runtime To compute the results of r matches, the basic scheme presented requires n PRF computations from both user (where n is the size of the contact list). The server does n hash table lookup. Assuming $O(1)$ for the lookup, all participants has to compute n computations. From the tests, the amount of time required in a high-end notebook is around 60ms.

Controlled leakage The leakage of the protocol in the server is limited to the size of the database and possible the number or records. This might be mitigated by adding dummy records to the hash table. This increases the size of the database and the size of the communication.

Hardware optimizations Modern processors have hardware acceleration and processor instructions for heavy cryptographic computations [jww16, iDW16]. The tendency in phone manufacturing is to increase the privacy, so implementing more hardware-accelerated cryptographic primitives seems like it will be the standard in the near future of mobile processors.

Other ideas Instead of using a simple hash table as database, it could be extended to other associated data such as an encrypted key in the case of multi-client SSE or timestamps for key revocation.

Some other elegant-but-yet-unpractical options would be using fully homomorphic encryption and/or oblivious RAM. Another alternative for the near-future is the Intel's new technology SGX, in which the processor contains a hard-coded secret key allowing the user to use the RAM while encrypted [Gue16].

Open problem As Moxie Marlinspike pointed out in [Mar14]: *For 10 million TextSecure (rebranded as Signal) users, a bloom filter would be around 40MB, requested from the server 116 times a second if every client refreshes it once a day.* This problem is still an open problem for big data sets.

References

[BDCOP04] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Advances in cryptology - eurocrypt 2004: International conference on the theory and applications of cryptographic techniques, interlaken, switzerland, may 2-6, 2004. proceedings. pages 506–522, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BF01] Dan Boneh and Matthew K. Franklin. Identity-based encryption from the weil pairing. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, pages 213–229, London, UK, UK, 2001. Springer-Verlag.

[BFP16] Raphael Bost, Pierre-Alain Fouque, and David Pointcheval. Verifiable dynamic symmetric searchable encryption: Optimality and forward security. *IACR Cryptology ePrint Archive*, 2016:62, 2016.

[CGKO06] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 79–88, New York, NY, USA, 2006. ACM.

[CH12] Philippe Camacho and Alejandro Hevia. Short transitive signatures for directed trees. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February*

27 - March 2, 2012. Proceedings, volume 7178 of *Lecture Notes in Computer Science*, pages 35–50. Springer, 2012.

[CJJ+14] David Cash, Joseph Jaeger, Stanislaw Jarecki, Charanjit S. Jutla, Hugo Krawczyk, Marcel-Catalin Rosu, and Michael Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. In *21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014*. The Internet Society, 2014.

[CJKT09] Emiliano De Cristofaro, Stanislaw Jarecki, Jihye Kim, and Gene Tsudik. Privacy-preserving policy-based information transfer. In Ian Goldberg and Mikhail J. Atallah, editors, *Privacy Enhancing Technologies, 9th International Symposium, PETS 2009, Seattle, WA, USA, August 5-7, 2009. Proceedings*, volume 5672 of *Lecture Notes in Computer Science*, pages 164–184. Springer, 2009.

[CT09] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear computational and bandwidth complexity. *IACR Cryptology ePrint Archive*, 2009:491, 2009.

[CT10] Emiliano De Cristofaro and Gene Tsudik. Practical private set intersection protocols with linear complexity. In Radu Sion, editor, *Financial Cryptography and Data Security, 14th International Conference, FC 2010, Tenerife, Canary Islands, January 25-28, 2010, Revised Selected Papers*, volume 6052 of *Lecture Notes in Computer Science*, pages 143–159. Springer, 2010.

[Gue16] Shay Gueron. A memory encryption engine suitable for general purpose processors. *IACR Cryptology ePrint Archive*, 2016:204, 2016.

[iDW16] iPhone Development Wiki. IOCryptoAcceleratorFamily. `http://iphonedevwiki.net/index.php/IOCryptoAcceleratorFamily`, 2016. [Online; accessed 24-May-2016].

[jww16] jww. Does iPhone support hardware-accelerated AES Encryption? - Stack overflow. `http://goo.gl/quF9ig`, 2016. [Online; accessed 24-May-2016].

[KMRS14] Seny Kamara, Payman Mohassel, Mariana Raykova, and Saeed Sadeghian. Financial cryptography and data security: 18th international conference, fc 2014, christ church, barbados, march 3-7, 2014, revised selected papers. chapter Scaling Private Set Intersection to Billion-Element Sets, pages 195–215. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.

[Mar14] Moxie Marlinspike. The Difficulty Of Private Contact Discovery. `https://whispersystems.org/blog/contact-discovery/`, 2014. [Online; accessed 23-May-2016].