Compromises in Private Set Intersection for Contact Discovery

Alì El Wahsh

ETH Zurich
elwahsha@student.ethz.ch

March 2, 2020

Abstract

In recent years a large pool of papers covering the use of private set intersection (PSI) for contact discovery was released. The security model considered for this problem assume a level of honesty from the server side, which is the party that benefit the most from acting dishonestly. In this report we compare one of the protocol proposed, the GC-PSI based on Garbled circuits and Cuckoo Filters, with a PSI protocol with stronger guarantees: the OPA which rely mostly on basic filed operations, and see if it is possible to have a viable secure alternative on mobile architectures. Under our analysis it appears that the overhead of the mutual security checks makes it harder on a communication cost level to implement a fully malicious secure protocol for the mobile market, and that redesigning existing protocols might not be the right road.

I. Introduction

Mobile messaging apps perform a particular operation after installation, called contact discovery, which allows a user to retrieve the subset of users from their address book that uses the same app. It is in the interests of the user to maintain their address book private and at the same time have a correct and timely execution of this operation. At the actual state, the most popular messaging app (Whatsapp) doesn't respect this privacy constraint, while some of its competitors are looking for viable private contact discovery.

In recent years, a large number of papers, proposing a solution for this issue, has been published. The focus of this report is one of this papers: *Mobile Private Contact Discovery at Scale* [4] by Kales et al. In their solution they proposed a Private Set Intersection(PSI) protocol that is viable on mobile architectures. While their performance results are promising, the security model used in the paper is quite weak (as we will see later). Their proposed model is assuming a semi-honesty from the server, which means that the server can be seen at worst as a passive attacker. This might be considered a legit assumption, since a malicious service provider would incur in legal issues and reputation damage, but this is beyond the scope of this report, that instead will try to see if it is possible to achieve similar results with a more secure model.

i. Contribution

While this report itself will not present any technique not already discussed in previous papers, it will bring light on some of them showcasing their strengths and weaknesses by comparing them. The main contribution would be showing an inverse correlation between performances and security. As a secondary contribution we will point out some possible improvements that can be adopted by both the protocols.

ii. Structure of the Report

In Section 2, we will explain the building blocks for both protocol with a focus on their security relevant properties.

In Section 3, we will present both protocols and their security model.
In Section 4 we will compare the protocols with a focus on the communication cost, and finally, in Section 5, we will draw our conclusion.

II. Background

In this section we will briefly introduce the primitives behind both the protocols under our analysis. For more insight on the optimizations used in the protocol, we invite you to read Section 3 of [4] and Section ADD of alge.

i. Private Set Intersection

Private set intersection is a cryptographic technique that allows two parties holding sets to compare encrypted versions of these sets in order to compute the intersection. In this scenario, neither party reveals anything to the counterparty except for the elements in the intersection.

Other variants of this exist, such as the Unbalanced PSI where one of the parties (usually the clients) hold a set with a much inferior number of elements than the other.

ii. Oblivious Transfer

Oblivious transfer (OT) is a cryptographic protocol that in its most basic form allows a sender P_1 to obliviously transfer one out of two messages (m_0, m_1) to a receiver P_2 based on a selection bit b chosen by P_2 s.t. P_1 learns nothing about b and P_2 learns only the corresponding message. It is usually implemented with a cryptographic primitive that allows the decryption of only one of the two messages. In our implementation we use the Bellare-Macali OT, where the messages are encrypted with elements of a cyclic group and the receiver has control only over one of the two elements.

iii. OPRF Evaluation

An oblivious pseudorandom function (OPRF) is a protocol between two parties: sender P_1 holding key k and receiver P_2 holding input x. After the invocation of the protocol, P_2 learns the output $f_k(x)$ of a keyed pseudorandom function (PRF) $f()$ .Additionally, it is guaranteed that P_1 does not learn anything about x and P_2 does not learn anything about k. It can be seen as the extension of an OT, even though it requires particular care, since in the case of an unbounded domain it is impossible to use the same techniques behind OT's implementations.

iv. Cuckoo Filters

Cuckoo filter are an alternative to the more popular Bloom filters. Like Bloom filters, they are a data structure for compact set representation that allows for fast membership testing with controllable false positive probability.

Cuckoo filters consist of a table of buckets with fixed bucket size b. Inside the buckets, so-called tags are stored. Tags are small bit strings obtained by hashing items. More precisely,to represent an item x in a Cuckoo filter, we first calculate its tag $t_x = H_t(x)$, where H_t is a hash function with output bit length v. This tag is stored in one out of two possible buckets. The position of the first possible bucket is calculated as $p_1 = H(x)$, where H is another hash function that maps the input to a position in the table of buckets. In case this bucket is already full, the tag is stored in the second possible bucket at position $p_2 = p_1 \oplus H(t_x)$. Note that it is always possible to determine the other candidate bucket p_j just from knowing its tag t_x and the current position $p_i : p_j = p_i H(t_x)$. If both buckets are full, one tag in the first is chosen at random, removed from that bucket, and moved to its other possible bucket. This procedure is repeated recursively until no more relocations are necessary.

v. Garbled Circuits

One of the most prominent techniques for secure two-party computation. (In the following the two parties are called garbler and evaluator.) The idea is to represent the function that is evaluated as a Boolean circuit and to replace each logical two-input gate by a garbled

gate. Each wire of the garbled gate is given two random wire labels, representing 0 and 1. To garble a gate, the garbler uses all four combinations of the gate's two input wire labels to encrypt the corresponding output wire label, based on the truth table of the original gate, and sends the resulting ciphertexts, the so-called garbled table, to the evaluator. The evaluator can then use the two input wire labels it possesses to decrypt one of the four ciphertexts and receive the output wire label, which is then used as input for subsequent gates. Since the garbler knows all wire labels, it can send the wire labels corresponding to its input bits to the evaluator. However, to ensure input privacy for the evaluator, the wire labels corresponding to the evaluator's input bits are retrieved via OTs. The garbler also sends information that allows the evaluator to decode the final output wire labels to 0 or 1.

vi. Oblivious Linear Evaluation

Intuitively, OLE is the generalization of OT to larger fields, i.e. it allows a sender and a receiver to compute a linear function $c(x) = ax + b$, where the sender holds a, b and the receiver inputs x and obtains c. OLE guarantees that the receiver learns nothing about a, b except for the result c, while the sender learns nothing about x. It is similar to the concept of OPRFE, aside the fact that we are working with any function over a possibly large finite field.

vii. Oblivious Polynomial Addition

An Oblivious Polynomial Addition is a cryptographic operation where two parties have access to a secret polynomial each and want to compute a polynomial C that contains the sum of the two polynomials (and by extension their roots). It can be implemented using OLE in this way:

Assume that the sender has an input polynomial a of degree 2d, and the receiver has a polynomial b of degree d. The sender additionally draws a uniformly random polynomial r of degree d. Both parties point-wise add and multiply their polynomials, i.e. they evaluate their polynomials over a set of $2d + 1$ distinct points $\alpha_1, ..., \alpha_{2d+1}$, resulting in $a_i = a(\alpha_i)$, $b_i = b(\alpha_i)$ and $r_i = r(\alpha_i)$ for i in [2d+ 1]. Then, for each of $2d + 1$ OLEs, the sender inputs r_i, a_i, while the receiver inputs b_i and thereby obtains $c_i = r_i * b_i + a_i$. The receiver interpolates the polynomial c from the $2d + 1$ (α_i, c_i) and outputs it.

III. Protocols under analysis

i. GC-PSI Protocol

The GC-PSI Protocol presented in [4] and illustrated in table 1 is divided in 4 phases where a server and a client try to perform contact discovery. These phases are:

- **Base phase**: The Server prepares the Garbled Circuits for a PRF and then proceeds to send the circuits to the Client.
- **Setup Phase**: The Server compute the PRF result for every single one of its elements and insert the result in a Cuckoo Filter. The (possibly large) Cuckoo filter is sent to the Client.
- **Online phase**: Using OPRF Evaluation over Garbled circuits, the Client computes the PRF of its elements with the Server and checks if there is a match inside the Cuckoo filter. If a match is found the item is considered element of the intersection
- **Update phase**: As new users join or leave the service, the Server send updates to the Cuckoo filter on the Client side in the form of insertion or deletion of elements.

ii. Security model

The protocol in [4] is described to be secure against a malicious Client, while assuming a semi honest Server. Under the protocol, the only attack that a client can mount is an enumeration attack and it can be prevented with a limitation over the queries a client can send to

Server		**Client**
Input: $X = \{x_1, ..., x_{N_s}\}$		Input: $Y = \{y_1, ..., y_{N_c}\}$
Output: $\perp$		Output: $X \cap Y$
	Base Phase	
Generate random PRF key k		
$\forall i \in [1...N_c]$		
$GC.Build(PRF_i)$	PRF_i	
	$\longrightarrow$	
	Setup Phase	
$\forall i \in users \quad compute \quad r_i = PRF(i)$		
$\forall r_i \quad CF.insert(r_i)$	CF	
	$\longrightarrow$	
	Online Phase	
$\forall i \in [1...N_c]$		$\forall i \in [1...N_c]$
	OTs for wire labels	
	$\longleftarrow$	
Send wire labels	$\longrightarrow$	
		If $CF.Contains(PRF_i(y_i))$
		$\rightarrow$put y_i in S
		Output S
	Update phase	
	Insert/delete N_u items	
$U = \emptyset$		
$\forall i \in [1...N_u]$		
$(t_i, p_i) = CF.find(PRF_k(u_i))$		
Put (p_i, t_i) in U	U, op $\in \{Insert, Delete\}$	
	$\longrightarrow$	
		$\forall i \in [1...N_u]$
		$CF.insert(t_i)$ or $CF.delete(t_i)$

Table 1: *Simplified GC-PSI protocol, Both parties start with a lists of users and at the end the Client will output the intersection. The server has the capability of building Garbled Circuits and Cuckoo Filters that will eventually send to the client.*

a server. Other than the previously mentioned attack, the authors gave an exhaustive explanation of the security measures used to avoid side channels attack (like an Interpolation Attack on the block cipher used as PRF). The real life scenario is a bit different and the main issue is to prevent the Server from learning the clients' elements.

Since the main actor in the protocol execution is the Server, it has to be assumed semi honest (it can only learn information from he correct execution of the protocol), that would not be enough if the Service Provider is interested in learning the contact list of the client (for example to learn their social net).

iii. Possible threats

In the case that the intersection is shared with the Service Provider, the Server can strategically insert or delete elements from he Cuckoo Filter allowing to increase the size of the intersection set S, so that it will cover all of the elements in the Client's address book. Another possible attack in this scenario requires to design the PRF to always output the same value

Server		**Client**
Input: $X = \{x_1, ..., x_{N_s}\}$	$W = \{w_1, ..., w_{N_s+1}\}$	Input: $Y = \{y_1, ..., y_{N_c}\}$
Output: $\perp$		Output: $X \cap Y$
Generate polynomial a:		*Generate polynomial* b:
$deg(a) = N_s$		$deg(b) = N_c$
$\forall x \in X \quad a(x) = 0$		$\forall y \in Y \quad b(y) = 0$
Generate random polynomial r :		
$deg(r) = N_s - N_c$		
Evaluate a *and* r *over* W :		*Evaluate* b *over* W :
$A = \{a(w_1)...\}$, $R = \{r(w_1)...\}$		$B = \{b(w_1)...\}$
$\forall i \in [1...N_s + 1]$		$\forall \in [1...N_s + 1]$
	OLE of $A_i + B_i * R_i$ $\longrightarrow$ $\longleftarrow$	
		With N_s *+ 1 points*
		interpolate $c(x) = a(x) + b(x) * r(x)$
		Output every y *that is root of* c

Table 2: *Simplified version of the OPA-based PSI protocol presented in [3]. It is also slightly adapted for the purposes of contact discovery. The degree of* r *is* $N_s - N_c$ *so that* $b * r$ *has degree* N_s.

(so much for the pseudo randomness) and insert this value in the Cuckoo Filter. The Client would output the entire contact list again.

iv. State of the art

As stated in [1] naiıve-hashing is the status quo for contact discovery, and as such a party learning the naive hashing of the elements of the other party is not to be considered an issue. Naturally an offline brute force attack would allow an adversary to learn the hashed elements and putting this in the context of contact discovery a server may learn all the numbers in a client address book. This takes us back to the original problem behind PSI for contact discovery: Protecting the client's privacy.

v. Algebraic approach to PSI

In [3], S. Ghosh and T. Nilges presented a PSI protocol with an interesting aspect to it: The protocol is secure in a fully malicious two-party setting. As we can see in Table 2 the basic idea behind it is:

- Both parties generate a polynomial over a finite field with their respective elements as roots, called a for the Server and b for the Client. Additionally, the Server generates a random polynomial r.
- All polynomials are evaluated over a known set of points.
- Server and Client initiate OLE where the Client obtains values $c_i = a_i + b_i * r_i$.
- The Client interpolates c from the points and check which of its elements are root of the new polynomial. In case of a positive match, the element is considered to be part of the intersection.

vi. Security model II

In the above protocol the main security concern is to avoid that none of the polynomials are equal to 0, otherwise the secrecy of the polynomial a or b would be compromised. To avoid such scenario, Ghosh and Nilges designed an enhanced OLE functionality. Assuming an OLE where we compute $a * x + b$ and we don't want to disclose b (as in the case a sender use input $x = 0$), it is required that the protocol cannot be forced to output b. To do so, b is multiplied with a correlated value via OLE and the right value for the linear function can only be retrieved with consistent inputs, while a cheating party would receive a uniformly random value.The same is applied in the case of a cheating Server ($a = 0$), which is our main concern in this model. For a more exhaustive explanation, please refer to Section 1 of [3].

vii. Possible threats II

There is the possibility that the Server may use a different polynomial a to try and learn other possible contacts other than the ones in the intersection, but the likelihood of finding the contact list (or even just an extra element) are abysmal. Even if the model is securer on paper it may still be possible to run side channels attacks, but this issue is highly correlated with the implementation of the protocol, so it is left as an open problem.

IV. Comparison

The GC-PSI protocol was fully implemented and tested on mobile architectures, while the OPA one was only designed on paper and it doesn't support Unbalanced PSI, so the protocol expects the two sets to be roughly of the same size. A good metric to evaluate these protocols is the communication cost, since, in the mobile connections context, the size of the data transfers is a critical aspect. Another fundamental aspect is the computational burden over the client side that is usually not as powerful as commonly used PCs.

i. Implementation of the OPA protocol

Since the aim of this report is to compare these different approaches and draw a conclusion on their usefulness, it seemed necessary to at least write a simulator for the later protocol. The goal of this simulator is to measure the key parameter of our analysis: the communication cost. The simulator for the OPA prototocol was entirely written in C++ (add link to code), using the NTL library that covers all the finite field arithmetic required by the protocol. While the computation is really executed by the simulator, for simplicity the computation cost is only computed over the size of the data, due the complexity of the project. The OLE used as base for the OPA is a specific variant presented in [2] called OLE with constant overhead.

ii. OLE with constant overhead

In order to achieve the results similar to the theoretical ones proposed by its creators, it was necessary to implement the OLE wit constant overhead. This variant requires only $O(1)$ OTs per round and it is obviously modelled according to the security model in [3]. This protocol requires as components a secret sharing scheme, a commitment scheme and noisy encoding, respectively implemented as Shamir's secret sharing, Pedersen commitment and the noisy encoding presented in [2]. A quick, and simplified, overview of the protocol itself is in table 3

iii. Comparison: Communication cost

As it can be see in 4 the difference in communication cost is way of the charts, as partially expected given the theoretical results of Ghosh and Nilges. The gargantuan cost of the OPA protocol is due to the necessity of numerous rounds of OLE run between Sender and Receiver. The number of rounds is directly proportional to the size of the bigger set (usually the Server in the contact discovery setting), so it is not possible to enhance the protocol for Unbalanced PSI, while in [4] the task was

Protocol Π_{OLE}

Let $P = \{\alpha_1, ..., \alpha_{(l+1)/2}, \beta_1, ..., \beta_n\}$ be a set of publicly known distinct points in $\mathbb{F}$. Let SS be a (ρ, n) secret sharing and COM be a commitment scheme.
Set $\rho = 3/4n$, $n = 8k$ and $l = n - \rho$.

1. Sender (Input a,b):
- Draw a polynomial $A \mid \forall \alpha_i \quad A(\alpha_i) = a_i$ and $B \mid \forall \beta_i \quad B(\beta_i) = a_i$
- Draw a random vector t
- Draw a random value e and computes $s \leftarrow SS.Share(e)$ and $(com, unv) \leftarrow COM.Commit(e)$
- Sends com to the receiver and run n OTs for (t_i, s_i)
2. Receiver (Input x):
- Encode x with noisy encoding and obtain (L, v), and interpolate X with points (β_i, v_i)
- Use bitmap L to execute the n OTs with Sender
- Obtain l values t_i and ρ values s_i. Compute $\tilde{e} = SS.Reconstruct(s)$
- Send $\tilde{e}$ to the sender
3. Sender:
- Check $e = \tilde{e}$. If positive send unv to the receiver
4. Receiver:
- Opens commitment and if it is correct sends v to the sender
5. Sender:
- Evaluates $A * v + B + t$ over all β_i and send the results to the receiver
6. Receiver:
- Subtract the known t_i from the respective received results and interpolates Y over the corrected results and β
A mutual correctness check and if it is positive the function can be evaluated from Y since $Y = A * X + B$ over α

Table 3: *Simplified Π_{OLE} protocol. k is the security parameter, aka the number of maximum OLEs that can be executed in a round.*

Protocol	N_s	N_c	Comm. cost
OPA [3]	2^{20}	2^{20}	$\sim 1.70GB$
GC-PSI [4]	2^{20}	1024	$\sim 30MB$
GC-PSI	2^{20}	1	$\sim 4MB$

Table 4: *Communication costs for the protocols under analysis, where N_s stands for the elements of the server, while N_c is number of elements on the client side.*

overcame with the use of compressed Cuckoo Filters, that allows to send encoded data for over a million of user in the small space of few MBs. The main culprit of the high cost for OPA are the large matrices used for the noisy encoding, which are required to be sent in every round. In the simulation no data compression technique was used to reduce this overhead, so it is still possible to get a better performance, but it may be still marginal.

iv. Comparison: Computational burden

The GC-PSI protocol is specifically optimized for mobile architecture and the heaviest cryptographic operations are handled by the ARM Cryptography Extension which adds native support for cryptographic primitives(ex. AES). It is available on most of the commercially available phones, but the adoption of the protocol would still mean a dropping support for a large pool of devices and it may be wise to wait. The

protocol is still not particularly demanding on the client thanks to the Base and Setup phases that are handled by the server. The OPA protocol is composed mostly of simple field operations (products and sums of elements) so by itself it is not very demanding. The secret sharing and commitment schemes can be picked from a large pool of secure and "light" options like Shamir's secret sharing scheme and Pedersen commitment. The use of optimized libraries (like NTL for C++) can reduce the execution of a round of Π_{OLE} to 40 ms (without taking in account the network overhead).

V. Conclusion

While the actual security model for private contact discovery is inadequate, there is no viable alternative with stronger security guarantees and high performance on mobile architecture. This is due to the extra steps a protocol has to take to ensure the correctness of the execution of the protocol. The model presented in [3] has proven to be easily implementable and performing well, but it's not a good fit for the mobile setting due to his high communication cost. The GC-PSI protocol was developed with specifically in mind the issues behind the limited resources of a mobile architecture and as it is possible to see from the results in [4] it is not too far from the bar set by Signal, and with a few tweaks it may be possible for it to reach real world deployment.
A fully maliciously secure and Unbalanced PSI protocol is still to be found, and it should be designed from the start as it, instead of tweaking or fixing a previous protocol. The requirement of having a number of operations that doesn't scale linearly with the server's set size and complete proof of correctness may be incompatible, but it still has to be proven.

VI. Acknowledgements

This report was written under the direct supervision of Professor Kenneth Paterson and with the help of (insert puvera anima) for proofreading it.

References

[1] Daniel Demmler et al. "PIR-PSI: Scaling Private Contact Discovery". In: *Proceedings on Privacy Enhancing Technologies* 2018 (Oct. 2018), pp. 159–178. DOI: 10.1515/popets-2018-0037.

[2] Satrajit Ghosh, Jesper Buus Nielsen, and Tobias Nilges. "Maliciously Secure Oblivious Linear Function Evaluation with Constant Overhead". In: *Advances in Cryptology – ASIACRYPT 2017*. Ed. by Tsuyoshi Takagi and Thomas Peyrin. Cham: Springer International Publishing, 2017, pp. 629–659. ISBN: 978-3-319-70694-8.

[3] Satrajit Ghosh and Tobias Nilges. "An Algebraic Approach to Maliciously Secure Private Set Intersection". In: *Advances in Cryptology – EUROCRYPT 2019*. Ed. by Yuval Ishai and Vincent Rijmen. Cham: Springer International Publishing, 2019, pp. 154–185. ISBN: 978-3-030-17659-4.

[4] Daniel Kales et al. "Mobile Private Contact Discovery at Scale". In: *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 1447–1464. ISBN: 978-1-939133-06-9. URL: https://www.usenix.org/conference/usenixsecurity19/presentation/kales.