Cryptographic Dating

John Mikhail, Emad Farag, Edgar Minasyan, Malek Ben Romdhane

May 2016

Abstract

Dating apps have proliferated recently, allowing clients to express interests in other clients. Many of these dating apps promise their clients to keep this data private unless there is mutual interest between two clients. When that happens, both clients are notified. However, at all steps, the server knows all the choices of the clients. We describe a cryptographic protocol for hosting a secure platform for dating, which hides any information about client choices from servers. Our protocol is not a direct peer-to-peer communication. It takes advantage of a centralized but oblivious server that acts as a middleware among clients. In this paper, we build on top of "Protocols for Secure Computations" by Andrew C. Yao. We briefly introduce Yao's method for secure multiparty computation then dive in the mathematical design of our protocol. Finally, we describe our proof-of-concept implementation of the protocol and we discuss the limitations it might have in a practical system.

1 Introduction

The dependence on technology in our everyday life has been exponentially increasing in the last decade. People use their smart-phones to rent cars, listen to music and stream movies. The emergence of technology did not just stop there. In the last five years we have witnessed large increase in the number of dating apps and their user base. The sensitive information these websites hold has raised privacy concerns about the online dating scene. In July 2015, Ashley Madison, a popular dating platform was hacked and information about millions of users were leaked online resulting in the breakup of many families. In this paper we tackle the issue of the security of dating platforms by proposing and building a cryptographically-secure dating service that imposes more security constraints than the ubiquitous platforms that are currently in use.

The dating problem is as follows: for every pair of people, each person must decide whether they like the other person or not. If both of them like each other, then both of them are informed about the result. If the feeling is not mutual, and one of the parties does not like the other person, they should obtain no information on whether the other person likes them or not. In our security model, we additionally impose the restriction that the platform maintainer (i.e. the server), does not learn anything about the choices of the clients.

The dating problem can be reduced to calculating an AND function where the two input bits represent the choices for each client. If both of them like each other, both will have 1 as their input bit, and the result will be 1. Otherwise, the result is 0. The goal is to allow two parties to compute the AND function together without compromising their inputs. If one client has 0 as their input, they should not know the other client's input bit. Most importantly, third parties involved in the computation should have no information about either the input or the result of the function.

This falls under a set of problems known in the world of cryptography as *secure multiparty computation*. There has been significant research on the subject, which served as basis for our solution to this problem. In this paper we outline previous work. We then explain our solution and the design choices behind it. We then present a summary of a proof-of-concept implementation. Finally, we discussed the limitations implied by our model.

2 Secure Multiparty Computation

The idea of *secure computation* was first introduced by Andrew C. Yao in 1982 [3]. The purpose of secure multiparty computation is to enable multiple distinct parties to securely compute a function, so that they would all learn the output of the function without learning anything about the other parties' inputs. There are several desired properties for a secure multi-party computation: privacy, correctness, guaranteed output delivery, fairness [2]. These are the main properties of an ideal security model, but some assumptions must be held to uphold all of the aforementioned properties. We firstly discuss the basic building blocks for the model and then describe the construction for secure multiparty computation. The case described is two-party computation, and one can find the general multiparty case in [2].

2.1 Oblivious Transfer

Oblivious transfer is a simple information transfer model involving two parties. The following version is suggested by Even et al [1]. The two parties involved are a sender and a receiver. As input the sender has a pair of strings (x_0, x_1) while the receiver has a bit $\sigma \in \{0, 1\}$. Oblivious transfer enables the receiver to obtain x_σ with the sender having no information about σ and the receiver having no information about the other string $x_{1-\sigma}$. The protocol guarantees the privacy mentioned above with the assumption of decisional Diffie-Hellman problem being hard.

2.2 Garbled Circuit

A garbled circuit is the main building block of our model. Any computation that can be performed by a Turing machine can be represented by a circuit of logical gates. Yao showed how to represent any such computation as a "garbled circuit" to able be able to use it in secure two-party computation. One party designs the circuit by representing each possible input bit in the circuit by a random bit string (symmetric encryption key). The garbled output of every gate is the original output encrypted using the keys representing its input bits.

For our case, our computation is only a single AND gate, $f(a, b) = a\ AND\ b$. The input bit

for Alice is represented by A_0 if she chooses 0 and A_1 if she chooses 1. Similarly, Bob's bit is represented by the keys B_0 and B_1 respectively. Moreover, the circuit has random bit strings for each of the possible outputs: R_0 and R_1. The garbled circuit for this function will have 4 garbled values - one for each possible input combination - encrypting the output bit string with the corresponding input symmetric encryption keys:

$$Enc_{A_0}(Enc_{B_0}(R_0))$$
$$Enc_{A_0}(Enc_{B_1}(R_0))$$
$$Enc_{A_1}(Enc_{B_0}(R_0))$$
$$Enc_{A_1}(Enc_{B_1}(R_1))$$

These 4 garbled values constitute the garbled circuit for two-party AND function.

2.3 The model

In the original formulation for the two-party computation model, Alice generates the garbled circuit, sends it to Bob along with the key representing her input bit. Afterwards Bob uses oblivious transfer to get the key representing his input bit and uses his and Alice's key to compute the output. That is, Bob chooses σ, then uses oblivious transfer to get B_σ from Alice without her knowing σ. Finally, Bob sends the computed answer to Alice and she informs about the result of the computation.

The problem with this approach is that it assumes that Alice and Bob are honest. Specifically, we assume that Bob will send the correct output he computed to Alice. However, if Bob is not honest, he can choose not to send the computed output to Alice, or send a different value. There is no way to enforce fairness in this model unless we assume that both parties follow the protocol without cheating.

In our design, Alice and Bob design the garbled circuit together and send it along with the keys representing their bits to a third-party server which will perform the computation and send the result back to both Alice and Bob. The server will not know the output of the circuit because it does not know which value of R represents 0 or 1. It also will not know what whether the keys for Alice and Bob represent 0 or 1.

3 Design

This section describes the different parties involved in our proposed solution to the dating problem, and how they interact. We base it on secure two-party computation.

3.1 Security Policy

The obvious approach to preventing a server from accessing private information is to remove the server from the process. According to Lindell and Pinkas, in order to achieve security against malicious adversaries, with fairness and guaranteed delivery, the number of corrupt parties t should be less than half of the number of parties n, i.e. $t < \frac{n}{2}$ [2, p. 84]. As two-party computation is used for our purposes, i.e. $n = 2$, the required level of security could be achieved if we have $t = 0$ corrupt parties. However, it is not reasonable to assume the honesty of both clients. Even though one can ignore $t = 2$ case, as it is not necessary to accommodate two clients trying to cheat each other, the system should still take care of $t = 1$ case.

Thus, to ensure fairness, we need to introduce a server that mediates the interaction between the two clients. This server makes sure that everybody is computing the same circuit, and that everybody proceeds to all steps at the same time, thus ensuring fairness. The server is also needed from a practical point of view to act as an intermediary of any communication between any two clients because it is unreasonable to expect all clients to be online at the same time. Thus, we need to make different assumptions about the different actors in this cryptosystem:

- **Server:** Since the server is here to ensure fairness, it should be honest-but-curious. It will follow the protocol but might be able to see any interaction. Given this information, it should still not be able to know any information about the choices of the clients and the final result, as this is the motivation of our project.
- **Clients:** In any dating service model, privacy is of at most importance. A pair of clients should only be notified of any result if they both like each other. Any notification should be symmetric. We assume that in a realistic model, many clients might want to cheat the system to figure out who likes them without expressing that they like the other person back. Thus we cannot assume an honest client. However, if both clients are cheating, then we do not make any guarantees. The system is only tasked with the protection of the privacy of honest clients.

3.2 Initial contact

Initially, the clients should establish a secure communication channel. Thus, we assume the existence of a trusted party to verify the identity of the clients. The clients use a certificate signed by the trusted party to authenticate themselves and all their communication. They initially register with the server. Then, two clients initiate a communication scheme. Initiating a communication scheme does not imply anything about the choice of a client. This communication scheme should be initiated with all members of the community or a randomized subset in order not to convey information. The server act as an intermediary for any communication between two clients.

3.3 Random Numbers Generation

As explained in the previous section the two clients need to come up with six random numbers together: A_0 and A_1 as the keys representing Alice's possible input bits, B_0 and B_1 representing Bob's possible input bits, R_0 and R_1 representing the output of the AND gate.

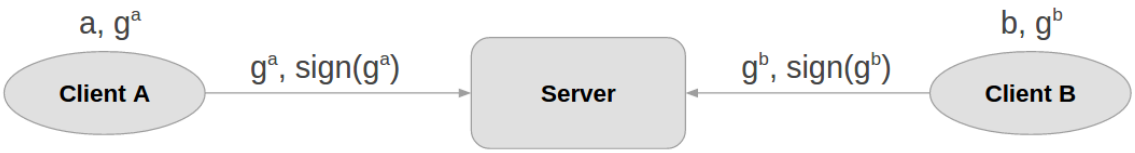

Figure 1: *Diffie Hellman key exchange to generate seed*

To hide the numbers from the server, Alice and Bob agree on a seed for a pseudo random number generation through Diffie Hellman key exchange. The Diffie Hellman key exchange is secure through the server since it guarantees security when there is a eavesdropper. We also sign the communication with the clients' certificates, which are signed by a trusted party. This prevents a man-in-the-middle attack. Once they both have a seed they generate six random numbers, using the first two random numbers to represent the possible bits of Alice's choice, the second two numbers to represent Bob's possible choice. The last two random numbers represent the AND gate output. The pseudo random number generator used should be cryptographically secure, so that the server cannot guess the seed or the generated numbers given a subset of those numbers. For the remainder of the paper, we will use the following names to refer to the randomly generated numbers:

$$\begin{gathered}
0 \text{ bit for Alice} \rightarrow A_0 \\
1 \text{ bit for Alice} \rightarrow A_1 \\
0 \text{ bit for Bob} \rightarrow B_0 \\
1 \text{ bit for Bob} \rightarrow B_1 \\
0 \text{ bit for Result} \rightarrow R_0 \\
1 \text{ bit for Result} \rightarrow R_1
\end{gathered}$$

3.4 Creating The Garbled Circuit

Using the generated random numbers, Alice and Bob encrypt R_0 and R_1 using the combination of (A_0, A_1) and (B_0, B_1) as keys to an industry standard symmetric encryption scheme (referred to here by Enc).

$$\begin{gathered}
Enc_{A_0}(Enc_{B_0}(R_0)) \\
Enc_{A_0}(Enc_{B_1}(R_0)) \\
Enc_{A_1}(Enc_{B_0}(R_0)) \\
Enc_{A_1}(Enc_{B_1}(R_1))
\end{gathered}$$

Alice and Bob then send these four ciphers along with R_0 and R_1 separately to the server. These 2 sets are sorted so that the server cannot guess which cipher belongs to which output or identify R_0, R_1. The server only proceeds if the two circuits Alice and Bob sent are the same, up to the byte level. This is guaranteed for honest clients because the numbers are the same, and they are sorted.

3.5 Sending Preferences

The server then asks both Alice and Bob to submit their preferences. Alice sends either A_0 or A_1 and Bob sends either B_0 and B_1. The server tries to decrypt all the ciphers it has. If one of the decryptions opens a number in (R_0 or R_1), the server sends that number to both Alice and Bob. If none of the decryptions is valid then the server will know that either Alice or Bob or both are trying to cheat.

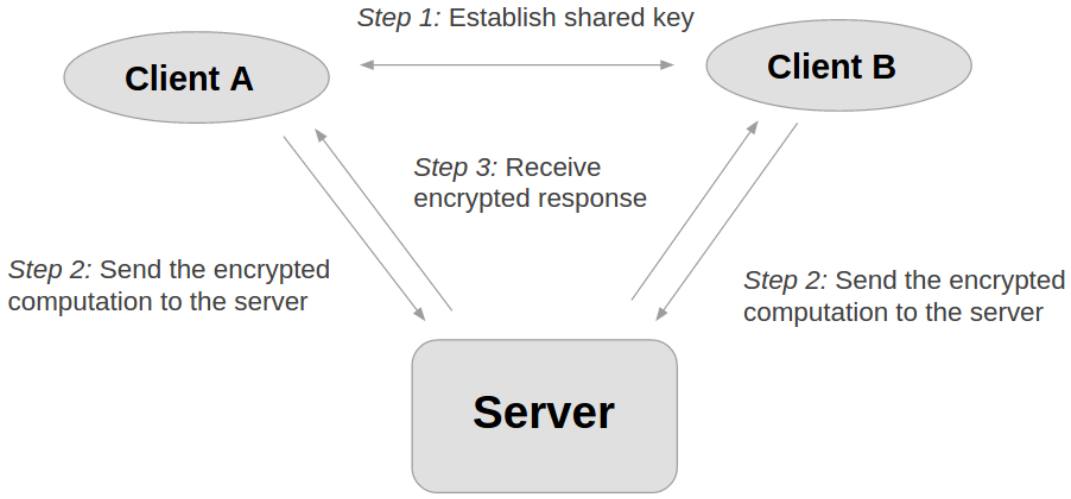

Figure 2: *Communication scheme between clients and server*

4 Implementation

A proof of concept system was implemented in Python. We used the cryptography[1] library for all base cryptographic functions and procedures, except for SHA256, which is already provided by Python. The Diffie-Hellman key exchange was done using the SECP256R1 elliptical curve, which is recommended by the National Institute for Standards and Technology. The pseudo random number generator was implemented using a counter mode block cipher using SHA256, where the IV is the RNG seed. AES is used for symmetric encryption. We use a 0 nonce because our keys are already randomized and used only once, so that would conserve IND-CCA2. The system functions correctly. A speed test run in a local commercial laptop running a local server/client interaction averaged about 22ms per client. Thus we believe that the system is efficient enough to be run in practical situations. The implementation can be viewed at `https://github.com/malekbr/crypto-dating`.

[1] `https://cryptography.io/`

5 Attack Model

- We proved that this system guarantees the privacy of the client's choices. With the assumption of an honest server, the server can perform no attacks as it has no access to any information about client's choices or the final result.
- The system guarantees privacy and fairness under the assumption that the information is not shared with the clients. Unless a malicious server is in place, this only happens in the case of hacking. Even in this case, a hacker cannot determine any information other than the choices of other people regarding himself. This is still a security improvement. Moreover, since the server does not know how to interpret the information, it has no incentive to keep it in memory. Thus a properly implemented system should have minimal impact in the case of a leak.
- Another possible attack is a corrupted client trying to get information about other parties' choices. In case of such an attack, we can assume that the adversary attacks an honest client. This means that the honest client follows the protocol and sends the correct circuit along with output bit strings. Therefore, any kind of modification to the circuit by the attacker will not be approved by the server and the process will stop. One way to figure out the choices of others for the attacker is to send 1 to everyone, and see all the matches, but with this attack as the other party also 'knows' that the attacker likes them, which just doesn't qualify as an attack.

6 System limitations

We introduce in this system an interactive communication scheme. This means that there needs to be synchronization, and the swipe and wait paradigm of modern dating apps does not apply anymore. Since there is a serve however, the clients need not to be connected at the same time, as communication can be buffered. Synchronization can thus be implemented reasonably.

In this system, in order to keep privacy, a client needs to express choices about some number of random clients, or about the whole community. This is to guarantee that the server does not figure out the interests from the established connections. This increases the communication complexity, and may put a limit on the total number of clients. This is more of an issue of scalability, but it can be solved in multiple ways. The program can be limited to fixed-sized communities. There can also be daily fixed random fixed size client pairing assignments. Some dating apps already employ such a system, so it would be easy to integrate.

7 Conclusion

We design in this paper a system that extends the privacy of the clients of dating apps. It blocks honest-but-curious servers from knowing the choices of the clients, including if two clients expressed mutual interest. The system is based on garbled circuits, and protects honest clients from cheating clients. Even in the case of a leak, the information leaked is minimal. The most information a person would know is who expressed interest in them. We also argue that the system is practical to implement, and provide a proof-of concept implementation.

References

[1] Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. *Commun. ACM*, 28(6):637–647, June 1985.

[2] Yehuda Lindell and Benny Pinkas. Secure multiparty computation for privacy-preserving data mining. Cryptology ePrint Archive, Report 2008/197, 2008. `http://eprint.iacr.org/`.

[3] Andrew C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.