Private Matchmaking Protocol with Security Enhancements

Marco Peraza, Varun Ravishanker and Cameron Setian

WHAT WE INVESTIGATED AND WHY:

The general matchmaking process involves assigning partners from a group of members based on selections each member makes in advance. Each member makes selections for desired partners and if two members select each other, they are matched. This general procedure can be useful in numerous different spaces, from online dating to job referral services. With the rise in popularity of these social networking applications and recent increase in concerns over software security and privacy, the development of a secure matchmaking protocol seems to be a topic worthy of investigation.

There are immediately several privacy concerns that arise when dealing with matching users based on their preferences. For example, it is likely that members want to keep their partner selections hidden for the selections they have not yet matched with. Members also may not wish for their commitments and/or their matches to be public knowledge. There is also concern over the trustworthiness of whatever third party is mediating the matchmaking process and whether it is using the information the members provide maliciously. There is even a potential concern of the third party snooping on the matches that members may want to be private.

Security is another key concern when implementing a social networking protocol like matchmaking. For example, a system could be vulnerable to attacks when the member is registering for matchmaking or selecting matches. Information about the member or his account could be compromised and an attacker could potentially make commitments on behalf of the member. If proper caution is not taken to secure any information that the third party stores, this database could also be vulnerable to spying o manipulation on the part of an attacker. Another key security concern for members would be the issue of repudiation once a match has been declared. In the extreme case, a member could cheat the system by matching every other member just to investigate who has selected them before revoking those matches.

WHAT OTHERS HAVE DONE:

Friendsy.net is a website that allows its users to indicate whether they would like to be friends, hook-up with, or date other users. Dartmouth College served as a test case for the site over 13X, when students were given cards with early access codes that allowed them to register on the website[2]. Registering for the website requires creating a profile. A profile must have at least a name, gender, and class year, but can also have a picture, major, home-town, and greek affiliation. A user can browse profiles and sort them by gender, graduation year, major, and greek affiliation. If user Alice selects that she would like to date user Bob, then Bob receives a notification that someone wants to date them. At this point, Alice is fully anonymous, because Bob only know's that *someone* wants to date him. Alice can lose some or all of her anonymity by sending hints to Bob such has her graduation year, gender, affiliation, major, or her name. If Bob selects that he wants to date Alice, then they are both notified that

they wish to date each other. All of the user's selections, successful or not, can be found on a page that only the user can access.

Dartmouth Last Chances was originally a matching site for graduating Dartmouth students to list other students that they would like to "hook-up with" before they graduate. Last year it was run by the Dartmouth Hacker Club and all students could use it. The web application used to develop an implementation of the protocol described in this paper is derived from this Dartmouth application. However, very few security measures were used in Last Chances. For example, there was no encryption when the user communicated with the server, so it was very vulnerable to MTIM attacks. The database was also unencrypted so this matchmaker relied solely on the trustworthiness of the third party. One of our group members was heavily involved in the development of Hacker Club's Last Chances.

Shin and Gligor[3] describe a privacy-focused matchmaking protocol and address many of the privacy concerns with Friendsy and Dartmouth Last Chances. Their protocol involves using a password-based authenticated key exchange(PAKE) boiled down into low-entropy secrets and add perfect blindness by replacing the username with a pseudonym, thereby adding security while also obscuring private details like a user's name and their wishes. This protocol however, does not address some other privacy concerns, such as the privacy of matches. Zhang and Needham[4] define the three main concerns for a private matchmaking protocol as secrecy of wishes or credentials, anonymity of users and authentication of matches. These are certainly privacy and security paradigms to be aware of when formulating a protocol, but there is still room for improving the security with more layering.

Lee and Kim[1] describe a secure matchmaking implementation using a simple five-step protocol that we used as a basis to model our protocol. This model utilizes ElGamal cryptography and Diffie-Hellman key agreement. In the first step, registration, a participant wishes to register with a trusted third party(TTP). This step utilizes ElGamal asymmetric cryptography. Using knowledge of a common multiplicative group G (likely provided by the TTP) with generator g, the user generates his private key priv in [1, order(G)] and uses this to compute his public key pub = g^d. The user then sends his name, public key with a certificate, and public key that the TTP then publishes this information on a "bulletin board". In the commitment phase, a user has the ability to select from any other registered user and makes a commitment by pulling the public key of the desired match. The user then generates a "couple ID" by combining his private key with his potential match's public key and hashes it, resulting in the couple ID $H(pub_{match}^{priv_{user}})$ and gives this information to the TTP. In the opening phase, the TTP uses a protocol for finding collisions in ElGamal ciphertexts to compute matches[1]. This essentially works by utilizing the Diffie-Hellman property that for users u_1, u_2, $H(pub_{u1}^{priv_{u2}}) = H(pub_{u2}^{priv_{u1}})$ so each user's couple ID can be checked against all others to find collisions easily. In step four, proof of coupling, the users release their private keys to prove that they did legitimately compute the couple IDs they posted. Finally, step five simply demonstrates verification by anyone of the coupling based on the mathematical computations in stages two, three and four.

WHAT WE DID:

The matchmaking protocol described in this paper has been optimized in several ways to improve upon privacy concerns present in the basic procedure while addressing some of the issues observed with the secure protocols discussed above. We also made some modifications to ensure that the app is usable.

We used a more robust asymmetric cipher, which utilizes elliptic curves rather than just discrete algorithms, like the ElGamal cipher used in Lee and Kim. Elliptic curve cryptography, ECC, for short, is thought to be more resilient than traditional asymmetric ciphers like RSA and ElGamal. RSA relies on the integer factorization problem, which has been repeatedly shown to be easier than first expected and requires very large keys for effective security. ElGamal is based on the discrete logarithm problem and is also less efficient than ECC in terms of security and required key size.

We wanted users to be able to use our application from the web browser without needing to download additional software or follow arcane series of steps. It would be infeasible to ask everyone to generate their own key pairs, upload their public keys, and load their private key into the browser's local memory each time that they want to interact with the application. Our solution to this problem was to generate the ECC key pair browser-side on the registration page, and then encrypt the private key with a user selected password before saving it to the server. The unencrypted private key never leaves the user's machine. This password essentially acts as the private key for the user and it is impossible for the private key to be compromised unless the user's web browser is compromised, or if the password and server-side database are compromised.

We realized that this scheme can also be applied to storing the names of other party associated with each commitment. If we stored the selected person's name in plaintext, the purpose of our cryptographic approach would be defeated. The server should not know what all of a user's preferences are, only the identities of matched couples. This makes it hard to present a UI for editing commitments, since the server only knows who made the commitment and the commitment's couple id. We added a column to the commitment table that stores the name of the selected person, but encrypted. The encryption and decryption both occur browser-side, as with the private key.

We started building off of Hacker Club's Last Chances application. We were able to reuse a lot of interface features, and almost all of the CSS and HTML. We substantially reengineered the backend, which uses the Flask web framework for Python, to use our cryptographic matching protocol. On the front end, we used the Stanford Javascript Cryptography Library (sjcl) to generate ECC key pairs, arrive at shared-secrets via Diffie-Hellman, and perform symmetric cryptography with the user's password.

WHAT WE LEARNED:

Over the course of developing this project, we learned several new security methodologies and also learned a lot about how many of the concepts we touched upon in class are actually implemented in

a software setting. With the goal of providing the utmost privacy to users of our matchmaker, the Diffie-Hellman anonymous key agreement protocol was certainly a major breakthrough for us. The ability to guarantee that a shared secret is private to the two parties who can compute it is exact function we needed to obscure knowledges of matches (and of non-matches) to the outside world (TTP included). Though we discussed it some in class, utilizing elliptic curve cryptography also with the Diffie-Hellman really helped to get a better understanding of how it works. Initially it seemed very daunting given the number of parameters that must be computed. Even when they were provided for us in the library, understanding how they interact and how to use the different parameters to generate our keys was certainly a learning experience.

In writing security-minded code, we also learned a lot about layering security. We found that the best approach to building the security of our application was simply adding more safety layers that must be bypassed in order to break it. We started getting better at seeing places that could be potential holes and fixing them. For example, we realized that by adding user convenience and generating keys for them, we also left a hole because we now have to store the private keys somewhere. By adding a second user password however, we were able to counteract some of this by symmetric encryption with the user password. We then were able to implement this same procedure to allow users to revoke their commitments before they have been matched much more quickly. In addition to the knowledge of finding holes, this project really elucidated Schell's argument about user experience generating potentially insecure, lazy code. Many times, making the process easier for the user involved layering more security to fill in the new holes.

WHAT SHOULD BE DONE NEXT:

We feel that we have established a very secure protocol to implement a private, secure matchmaking application. One area that could stand to be improved, and that Lee and Kim[1] included in their protocol is more widespread verification. In the current system, the TTP declares matches and there is no method for proof of coupling (at least not one mediated by the application) or for providing anyone with validation. While this may be useful to have, since we perform most user computation for him on the client side, this may not be as big of a concern for users.

Another area for furthering our implementation could be making the protocol more robust to various types of matchmaking. As is, the protocol only takes a name as information, but for applications like job referrals much more information would likely be required. This would again open up more holes simply because we must store more data, but would also add to the user experience. We also did not improve on the security of communication channels between the user and the server. While in theory it's not feasible to generate any information about users from the information we send back and forth, the code could still be succeptible to web attacks. Therefore, an additional step to improving security of this protocol would be to incorporate a communication protocol like SSL for communication between the client and the server.

References:

[1] Byoungheon Lee and Kwangjo Kim. “Secure Matchmaking Protocol”. *Information and Communications University*.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.22.2469&rep=rep1&type=pdf

[2] http://www.dartbeat.com/2013/07/02/friendsy-fosters-student-relationships/

[3] Ji Sun Shin and Virgil D. Gligor. “A New Privacy-Enhanced Matchmaking Protocol”. *University of Maryland and Carnegie Mellon University*.
https://www.usukita.org/papers/3353/PrivacyEnhancedMM.pdf

[4] Kan Zhang and Roger Needham. “A Private Matchmaking Protocol”. *Cambridge University Lab*.
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.54.835&rep=rep1&type=pdf