Solving the Dating Problem with the SENPAI Protocol

Abstract

The SENPAI protocol (Secure ENcrypted Protocol for Affection Information protocol) builds on work by [1] to allow efficient secure two-party computation on a problem of general interest with security against covert adversaries, while avoiding the overhead of zero-knowledge proofs. We will discuss historical attempts to solve the problem under discussion, followed by an explanation of the SENPAI protocol and its security properties.

1. Introduction

The Dating Problem is an extremely difficult computational problem that has plagued humanity since time immemorial. The problem concerns two agents Alice and Bob (or Alex and Bob, or Alice and Beatrice), each of whom may or may not have a crush on the other. Each is initially unaware of the other's feelings, and if they have a crush, they would like to know whether the other does as well; however, each would like to reveal their crush only if the other shares the interest. This problem is of widespread interest and applicability; for example, a solution to the Dating Problem is a hitherto unremarked-on prerequisite for the Stable Marriage Algorithm.

This problem, once thought intractable, can in fact be solved efficiently using cryptographic methods. A protocol providing security against semi-honest adversaries was presented as early as 2008 in a paper by Scott Aaronson. In this paper, Aaronson suggests that zero-knowledge proofs could be used to remove the semi-honesty assumption; however, this poses serious challenges for any practical implementation of the protocol. We have now augmented Aaronson's protocol with a few extra protections to create the SENPAI protocol, which is efficient, easily-implemented, and maximally resistant against covert adversaries.

In Section 2, we briefly outline historical solutions to the Dating Problem, and discuss their advantages and disadvantages. In Section 3, we present Aaronson's protocol and the SENPAI protocol, along with an analysis of its security properties. In Section 4, we discuss extensions of the protocol and possibilities for future work.

2. Historical Approaches

Dating has been a difficult problem for people and cultures throughout the centuries; many, many methods have been proposed as solutions. We will offer a sample of past attempts before explaining and demonstrating SENPAI.

2.1 The Arranged Marriage Protocol

The Arranged Marriage Protocol (AMP) is one of the oldest solutions to the dating problem, perhaps older than even cryptography. AMP is explained in detail in [3], but we will briefly outline the protocol here.

1. All participants agree to give up on finding a party that they have an interest in.
2. Each participant designates a neutral party (a "parent") to negotiate on their behalf and arbitrarily decide on pairings.
3. Each participant is matched with an equally unsatisfying partner.

This protocol, while guaranteeing perfect security (since no participant's actual preferences are involved at any stage), is somewhat lacking in terms of effectiveness; the probability of obtaining the desired result drops rapidly as community size increases, and as wealth of one's actual desired partner decreases.

2.2 The Just Man Up And Ask Protocol

The Just Man Up And Ask Protocol (JMAP) is often proposed as a naïve solution to the Dating Problem. In JMAP, if Alice has a crush on Bob, she simply *tells him*, and asks for his response. This protocol is certainly effective in the case where both parties' responses are Yes; they will learn the other's response quickly and efficiently, with a minimum of overhead and no third-party involvement. However, if Bob's answer is No, Alice has now unnecessarily leaked her Yes response to Bob. This scenario leads to non-negligible awkwardness for both parties with high probability, and thus the degree of risk posed by this protocol is clearly unacceptable.

There is also the issue that the JMAP protocol is fundamentally asymmetrical, posing greater risks for the initiator than for the recipient. Thus, there is not necessarily any good way to decide who should initiate a JMAP exchange, as both parties would prefer not to be the initiator. This has historically often been resolved by widespread standards designating all agents as either initiators or recipients. However, this approach breaks down when both parties are designated recipients; this is known as the Lesbian Sheep Problem. For further information, see [7].

2.3 The Meddling Friend Protocol

The Meddling Friend Protocol (MFP) protocol consists of Alice and Bob each revealing their preferences to a trusted third party, Trent, who then informs them if they have a mutual crush. [10] provides one of many sample implementations.

Unfortunately, this protocol is completely dependent on the trustworthiness of Trent, as they receive all of Alice and Bob's information and are entirely free to manipulate the results of the protocol. If Trent has been compromised, perhaps by a nation-state adversary, Alice and Bob's love life will be completely in the hands of a malicious attacker. Thus, unless Alice and Bob can find a third party who they both trust completely, the security of this protocol is quite poor.

2.4 The Start Out As Friends Protocol

The Start Out As Friends Protocol (SOAP), as presented by [9], is summarized in Figure 1. In this protocol, Alice and Bob participate in many exchanges over a long period, transferring pieces of information that provide only incremental evidence of their responses. Over time, each party can become more and more confident in the other's answer, until the probability of a mismatch is low enough that a JMAP exchange becomes feasible.

Unfortunately, the runtime of this protocol may be inordinately long in some cases, and deriving evidence from the individual transmissions can be difficult. There are also highly complex failure modes when dealing with a malicious adversary, as Munroe illustrates in his paper.

2.5 The You've Got Mail Protocol

The You've Got Mail Protocol (YGMP), based on original work by [8] and refined by [4], involves Alice and Bob performing a SOAP/JMAP exchange over a pseudonymous channel, and then revealing their actual identities to each other if the JMAP is successful. The pseudonymity eliminates many of the risks of JMAP, and reduces the problem to simply confirming each other's actual identities, which can

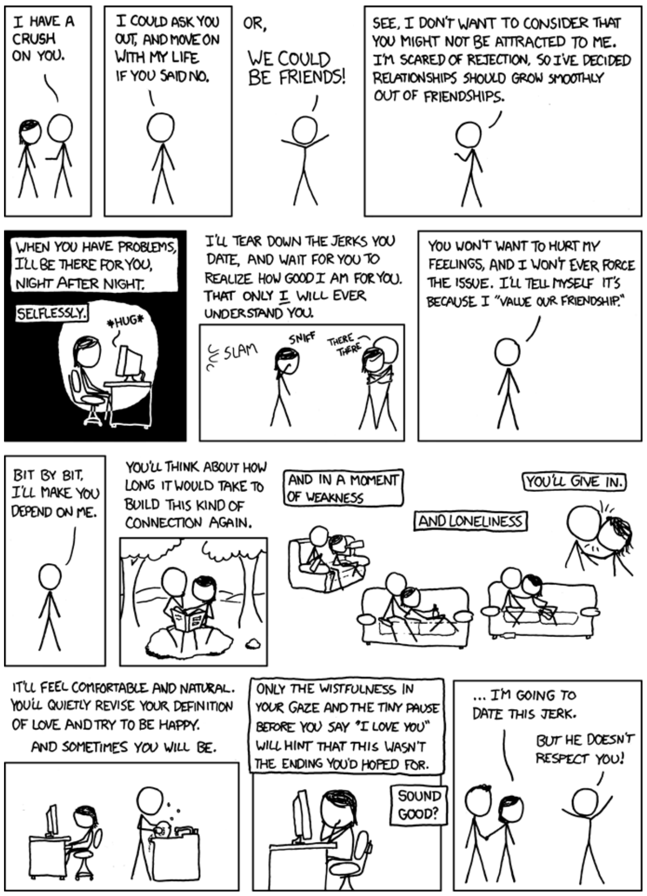
I HAVE A CRUSH ON YOU.
I COULD ASK YOU OUT, AND MOVE ON WITH MY LIFE IF YOU SAID NO.
OR,
WE COULD BE FRIENDS!
SEE, I DON'T WANT TO CONSIDER THAT YOU MIGHT NOT BE ATTRACTED TO ME. I'M SCARED OF REJECTION, SO I'VE DECIDED RELATIONSHIPS SHOULD GROW SMOOTHLY OUT OF FRIENDSHIPS.
WHEN YOU HAVE PROBLEMS, I'LL BE THERE FOR YOU, NIGHT AFTER NIGHT.
SELFLESSLY.
HUG
I'LL TEAR DOWN THE JERKS YOU DATE, AND WAIT FOR YOU TO REALIZE HOW GOOD I AM FOR YOU. THAT ONLY I WILL EVER UNDERSTAND YOU.
SLAM
SNIFF
THERE THERE
YOU WON'T WANT TO HURT MY FEELINGS, AND I WON'T EVER FORCE THE ISSUE. I'LL TELL MYSELF IT'S BECAUSE I "VALUE OUR FRIENDSHIP."
BIT BY BIT, I'LL MAKE YOU DEPEND ON ME.
YOU'LL THINK ABOUT HOW LONG IT WOULD TAKE TO BUILD THIS KIND OF CONNECTION AGAIN.
AND IN A MOMENT OF WEAKNESS
AND LONELINESS
YOU'LL GIVE IN.
IT'LL FEEL COMFORTABLE AND NATURAL. YOU'LL QUIETLY REVISE YOUR DEFINITION OF LOVE AND TRY TO BE HAPPY.
AND SOMETIMES YOU WILL BE.
ONLY THE WISTFULNESS IN YOUR GAZE AND THE TINY PAUSE BEFORE YOU SAY "I LOVE YOU" WILL HINT THAT THIS WASN'T THE ENDING YOU'D HOPED FOR.
SOUND GOOD?
... I'M GOING TO DATE THIS JERK.
BUT HE DOESN'T RESPECT YOU!

Figure 1. SOAP and its failure modes

be done safely without leaking any sensitive information in and of itself.

However, YGMP introduces many new issues. Due to the necessity of pseudonymity, it is difficult to ensure with any reasonable probability that one is carrying out the protocol with the correct partner, and of course an MITM attack (Matchmaker-In-The-Middle) is almost impossible to prevent.

Furthermore, if Alice is interested in both Bob and Clara, conducts YGMP exchanges with each of them, and receives a Yes from Bob's pseudonym and a No from Clara's pseudonym, she is now unable to reveal her identity, as she has no way of knowing whether she's revealing herself to the person she received a No from.

All in all, the YGMP, while it provides a clever workaround to some of the difficulties of SOAP/JMAP, is too problematic in many other respects.

2.6 The Shoe Locker Protocol

The Shoe Locker Protocol (SLP) has been investigated extensively by Japanese researchers. SLP is in some sense a compromise between JMAP and YGMP. It proceeds similarly to JMAP, except that the initiator communicates using a pseudonym, revealing their identity only if they receive a favorable answer. That is, if Alice is the initiator, she sends Bob a pseudonymous message declaring her Yes response. Bob then responds, and learns Alice's identity if and only if his answer is Yes. (This is commonly accomplished by instructing the recipient to come to a specific meeting place if their answer is Yes, where the initiator is waiting to reveal themselves.)

Reducing the pseudonymity to one side of the exchange removes some of the problems inherent in YGMP, while still avoiding some of the risks of JMAP. However, while the initiator knows the recipient's identity, it is vital for this protocol that the recipient be initially unaware of the initiator's identity. Thus, the recipient may respond favorably, only to find that the initiator was not the person they expected, resulting in the same failure mode as standard JMAP. A malicious initiator could even refuse to reveal their identity after learning the recipient's reply, resulting in some of the recipient's information being leaked. Furthermore, as noted by [5], a security-conscious recipient may react extremely poorly to an attempted SLP exchange.

3. The SENPAI Protocol

Having examined historical protocols for resolving the Dating Problem, we find that they all fail in one way or another at the basic goal of the problem: Alice and Bob should each learn the logical AND of their responses, *and nobody should learn any other information.* In particular, Alice should learn Bob's response only if her own response is Yes, and vice versa; and no third parties should learn anything about Alice and Bob's responses. In this section, we present the protocol of [1], which achieves this goal for semi-honest participants. We then examine the ways in which dishonest participants can cheat, and present our SENPAI protocol, which builds on Aaronson's protocol to provide maximal security against covert attackers.

3.1 Aaronson's Protocol

Aaronson's protocol consists of the following steps:

1. Alice generates an RSA modulus N, whose factors she knows, and reveals N to Bob.
2. Alice generates two residues x and y modulo N. x consists of 0 (with some suitable padding scheme applied), and y consists of Alice's response (0 if her response is No, 1 if her response is Yes).
3. Alice takes x^3 and y^3 modulo N, and sends both of these to Bob.
4. Bob chooses a random residue r modulo N, computes x^3r^3 or y^3r^3 modulo N depending on his response (using x if his response is No, y if his response is Yes), and sends the result back to Alice.
5. Alice takes the cube root modulo N of the value she receives, producing either xr or yr, and sends the result back to Bob.
6. Bob divides by r, learning the logical AND of their responses, and shares the result with Alice.

3.2 Analysis of Aaronson's Protocol

It is easy to see that Aaronson's protocol functions correctly when conducted between semi-honest parties, who will try to learn as much as they can but will carry out the protocol correctly. Bob, not knowing Alice's RSA private key, is unable to take cube roots modulo N, so he can't learn y from Alice's initial message. Then, Alice learns either xr or yr, but not knowing r, she is unable to distinguish these, so she doesn't learn Bob's response. Finally, Bob learns the value of whichever of x or y he chose; thus, he will get a 1 if and only if Alice put a 1 in y (her response was Yes), and he chose y (his response was Yes). Therefore, the protocol correctly reveals the AND of Alice and Bob's responses and no other information, without any third parties involved.

However, since Bob learns the result before Alice, it is absolutely trivial for Bob to cheat, learning Alice's response regardless of either of their responses. Bob simply needs to input a 1 into the protocol (choose y), and then at the end, report to Alice that the result of the protocol was a 0, regardless of the true result. Since Bob inputted a 1, he learns Alice's response; but Alice always receives the result she would get if Bob put in a 0, so she doesn't learn his true response.

3.3 Motivation for the SENPAI Protocol

We have set out to design a protocol that eliminates these avenues for cheating. Alice must learn Bob's response if and only if her response is Yes, and vice versa, with neither of them learning any other information, even if one of them violates the rules of the protocol. In his presentation of the original protocol, Aaronson suggests that zero-knowledge proofs can be used to prevent cheating. However, we find this approach unsatisfying, partly because zero-knowledge proofs would make the protocol much more unwieldy and time-consuming, and partly because we have no clue how to actually implement ZKP in this context. We would like to find a simpler, more efficient way to augment Aaronson's protocol to prevent cheating. First, though, we will address exactly what kinds of cheating it's possible to prevent, and what kinds it makes sense to prevent.

We note that no matter how a dating protocol is constructed, it can at best provide security against covert adversaries; that is, adversaries who are willing to cheat only if they won't get caught. An adversary who is willing to get caught cheating is unstoppable: They can simply input a Yes, perform the protocol honestly (learning the other person's response), and then if the result is Yes, simply "take back" their Yes response. Thus, we will only aspire to prevent *undetectable* cheating, since that's ultimately all we can do.

Also, we observe that even if Bob violates Aaronson's protocol as we described above, this is only *useful* to Bob under certain circumstances. In particular, if the true result of the protocol is a 0, there is absolutely no reason for Bob to tell Alice it was a 1: If Alice's bit was 0, then she catches Bob cheating, and if Alice's bit was 1, then this is equivalent to Bob just submitting a 1 and performing the protocol honestly. Thus, there's no need to detect fake 1s sent by Bob; we just need to detect fake 0s. In other words, we just need a way for Bob to prove that the output of the protocol was really a 0. We now present our modified SENPAI protocol, which neatly accomplishes this goal, preventing any undetectable cheating on the part of either participant.

3.4 The Protocol

The SENPAI protocol proceeds along more or less the same lines as Aaronson's protocol. However, before generating x and y, Alice chooses a random bitstring s, long enough that it can't be reasonably guessed. Then, x consists of 0 *and* s (with some padding scheme), and if y is 0, then y likewise consists of 0 and s. s thus becomes a sort of *certificate*, proving that one has somehow obtained a 0. For reasons that will become clear shortly, Alice commits to s with a bit-commitment protocol, and sends the commitment to Bob. The protocol proceeds as normal, and if Bob gets a 0 at the end, he will then also have s. In this case, he verifies that s matches the commitment, and then sends s back to Alice, proving that the 0 is genuine. Finally, regardless of the result, Alice reveals x to Bob (again, we will explain soon why this is necessary); Bob checks that this x contains an s matching the committed value, and that its cube modulo N is the value for x^3 he was given earlier.

3.5 Analysis

We now see that Bob can only learn s by learning the value of x, or of y if y contains 0. Since Alice won't accept a 0 result without Bob showing her s, it is now impossible for Bob to cheat if Alice carries out her side of the protocol correctly.

However, if we aren't careful, it's now possible for Alice to learn Bob's response while appearing to input a 0: she can put s into x and some $s' \neq s$ into y, and then determine Bob's response by seeing which value he sends back. Having Alice commit to s and Bob verify x ensures that if Alice wishes to cheat in this way without Bob catching her, she'll need to put the s she committed to in x. Then, if Bob's response is 0, she'll learn that it's 0, and Bob will be none the wiser; but if Bob's response is 1, he'll discover that the s' in y doesn't match the value Alice committed to, so he'll catch her cheating. We thus see that this is equivalent to Alice inputting a 1, playing the protocol honestly, and then "changing her mind" if Bob's response is also 1; since, as we've said, it's impossible to prevent this attack, our algorithm provides the maximum possible security against covert adversaries.

4. Further Work

4.1 Ternary Responses

Thus far in this paper, we have assumed that the presence or absence of a crush is a simple, binary phenomenon: Either Alice is interested in Bob, or she isn't. However, this fails to account for some of the nuance of real-world relationships; it may be the case that Alice would be open to dating Bob if he asked her, but wouldn't proactively pursue a relationship with Bob otherwise. In other words, a more complete model would include a "Maybe" answer in addition to Yes and No; we would then like our protocol to return a 1 if one participant has a Yes and the other has at least a Maybe, and return a 0 otherwise.

We claim that this can be accomplished with a fairly simple modification to the SENPAI protocol. Alice simply needs to generate 3 values instead of 2; call these x, y, and z. These will be set to $0, 0, 0$ if Alice's answer is No, $0, 0, 1$ if her answer is Maybe, and $0, 1, 1$ if her answer is Yes; as before, the 0 values will also include a certificate s which Alice chooses and commits to. The protocol proceeds analogously, with Bob choosing one of x^3, y^3, or z^3 depending on his answer, and ultimately learning x, y, or z, once again verifying s and x. We henceforth refer to this augmented protocol as SENPAI-MTT (More Than Two).

Unfortunately, SENPAI-MTT does not have quite the same security properties as the original SENPAI protocol. It is possible for Alice to submit all 0s, putting s into x and z but putting some other s' into y. In this case, Alice is able to determine whether Bob's answer is a Maybe, and will only get caught cheating if this is indeed the case. Unlike similar potential attacks with the original SENPAI protocol, there is no equivalent attack in which Alice carries out the protocol honestly and "changes her mind" if necessary; this is a genuine violation of our requested property of perfect covert security. We note that this is not a very powerful attack; worst-case, Alice secretly learns that Bob's answer is either Yes or No, but not which one. We believe that this is the only practical attack on SENPAI-MTT; users should exercise their own judgment as to whether the potential information leakage is acceptable.

4.2 Practical Issues

As is noted in [1], if Alice simply suggests to Bob that they carry out a SENPAI exchange, this in and of itself is indicative of interest on Alice's part. Thus, the only non-awkward way to actually use the SENPAI protocol is to assemble large groups of people, and have every pair of people carry out the protocol.

The ideal situation would be to have *everyone* (for some definition of everyone) perform SENPAI exchanges in this fashion. This minimizes the amount of metadata leakage relating to participants' social networks, and maximizes the likelihood that participants will actually have a chance to conduct the protocol with their crushes. However, in a group of n people, this requires every single participant to perform $O(n)$ work, regardless of the number of people they are actually interested in. Also, unless everyone in the group is online at the same time, a server may need to act as an intermediary, storing intermediate transmissions to be passed on to clients at the next opportunity.

As the size of "everyone" grows, all of this eventually becomes unfeasible. This could be resolved if there were some way of only performing computation for one's actual crushes; the work of [2] on covert two-party computation may offer a solution, but their paper was kind of confusing and we only skimmed it. For now, our recommended usage of the SENPAI protocol is for groups of 10–20 friends to organize parties where people will gather at a specific time and conduct the protocol among all pairs of attendees.

An unavoidable issue with group SENPAI exchanges is the problem of love triangles. If Alice, Bob, and Charlie are all part of a SENPAI party, and it turns out that Alice and Charlie are both interested in Bob and Bob is interested in both of them, awkwardness ensues. The authors' recommended solution is polyamory; in cases where this is not possible, another option is for each person to conduct their SENPAI exchanges serially, and after a successful exchange, change their answers for all of the remaining partygoers to No.

However, this means that the order in which exchanges take place can now influence the results; Alice now needs to order the other participants by how attracted she is to each of them, and conduct her exchanges in that order. This can potentially result in unsatisfiable ordering constraints: If Alice, Bob, and Charlie are all interested in each other, but Alice prefers Bob to Charlie, Bob prefers Charlie to Alice, and Charlie prefers Alice to Bob, there's no ordering of SENPAI exchanges that can satisfy everyone's preferences. We remark that this situation is just generally shitty, and seriously, what do you expect us to even do here? We observe that no stable relationship exists in this scenario, and reiterate our recommendation of polyamory to resolve issues with group SENPAI exchanges. In practice, the ordering of exchanges can be determined by which participants are quickest to select their preferences and initiate the protocol; in other words, first come first served.

4.3 Quantum Adversaries

Because the SENPAI protocol as presented above relies heavily on RSA encryption, it is of course possible for Bob to cheat if he has access to a quantum computer: he can simply use the work of [11] to obtain y from y^3, and immediately learn Alice's input before he has even chosen his own input. Fortunately, the particular choice of RSA is not crucial to the protocol; any partially homomorphic cryptosystem will work equally well. The key feature of RSA is that Bob can manipulate the encrypted values of x and y in such a way that Alice can decrypt them to obtain values that tell her nothing, but Bob can then extract the original x or y from those decrypted values. Thus, a future version of the SENPAI protocol might use a lattice-based cryptosystem such as that of [6] to provide quantum-resistant homomorphic encryption.

5. Conclusion

We believe that the SENPAI protocol is a significant improvement on the dating protocols commonly in use today, and is more simple and practical to implement than the original protocol proposed by [1]. We hope to see significant adoption of the SENPAI protocol once a few kinks are ironed out and an efficient, secure implementation is achieved.

References

[1] S. Aaronson. 6.080/6.089 Great Ideas in Theoretical Computer Science. MIT, Cambridge, Massachusetts, 2008.

[2] L. von Ahn, N. J. Hopper, J. Langford. Covert Two-Party Computation. In 37th STOC, pp. 513–522, ACM Press, 2005.

[3] J. Bock, S. Harnick, J. Stein. Fiddler on the Roof. 1964.

[4] D. Ephron, N. Ephron, T. Hanks, M. Ryan. You've Got Mail. Warner Bros. Entertainment Inc., Burbank, California, 1998.

[5] S. Gatoh, Y. Takemoto. Full Metal Panic? Fumoffu. Fuji Television, Tokyo, Japan, 2003.

[6] C. Gentry. Fully homomorphic encryption using ideal lattices. Symposium on the Theory of Computing (STOC), 2009, pp. 169-178.

[7] H. A. Landman. The Problem of Lesbian Sheep. Fort Collins, January 2004. http://www.polyamory.org/~howard/Poetry/lesbian_sheep.html.

[8] M. László. Parfumerie. Pest Theatre, Budapest, Hungary, 1937.

[9] R. Munroe. Friends. May 2008. https://xkcd.com/513/.

[10] W. Shakespeare. Much Ado About Nothing. London, 1623.

[11] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. arXiv:quant-ph/9508027v2.