Literature Review

VE450 TEAM 20

1. Problem Clarification

Our target is to build up a dating system, where users can commit 'interested'/'uninterested' to others. The system will only match and inform two users when they are exactly interested in each other. A few constraints need to be accomplished:

- Even though server side may help match two users, the server itself will not have any idea bout this private information.
- To avoid embarrassment case that might happen in reality, if a user A is not interested in another user B, then A should not know whether B is interested in A in any cases.

2. Related Works/Problems and Their Solutions

2.1. Cryptographic Dating

2.1.1. Overview

The dating problem can be reduced to calculating an AND function. If on client has 0 as the input, he or she should not know the other client's input. Also, according to our demand for the dating system, we require the server not learning anything about the choices of the clients. However, we cannot just remove the server from the process because it would cause fairness problem. So, method of secure multi-party computation is needed.

The main idea of the approach is that a circuit of logical gates can represent any computation that can be performed by a Turing machine, and such circuit is called "garbled circuit". In our case, the computation is $f(a, b) = a \wedge b$. We can represent Alice's input bit by A_0 and A_1 for 0 and 1 correspondingly. Similarly, Bob's bit is represented by B_0 and B_1, the possible outputs are represented by R_0 and R_1.

2.1.2. Initial contact

Initially, Alice and Bob should establish a secure communication channel. This communication scheme should be initiated with all members of the community

or a randomized subset in order not to leak information. The server then could act as an intermediary for any communication between two clients.

(*However, in this way, I don't know if it is possible to add a new client after the scheme was established.*)

2.1.3. Random numbers generation

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 seed then is used to generate 6 random numbers, the first two are used to represent the possible bits of Alice's choices, the second for Bob, and the last two for the results. Also, the pseudo random number generator used should be cryptographically secure (2^{128}).

2.1.4. Creating the garbled circuit

Using the generated random numbers, Alice and Bod encrypt R_0 and R_1 using the combination of $(A0, A_1)$ and (B_0, B_1) as keys to an industry standard symmetric encryption scheme (referred by Enc).

(*Maybe AES could be used here.*)

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

Alice and Bob then send these four ciphers along with R_0 and R_1 separately to the server. The sever only proceeds if the two circuits are the same.

(*The sets Alice and Bob send to the server should be sorted so that the server cannot guess which cipher, however, it is not clear for me how to sort.*)

2.1.5. Sending preferences

After all the above processes are done, the server asks Alice and Bob to submit their preferences. The server tries to decrypt all the ciphers it has and if one of the decryptions opens a number in R_0 or R_1, the server sends that number to both Alice and Bob.

2.2. Bit Commitment

2.2.1. Collision-free hashing

Assumption The Sender is computationally bounded and the Receiver is all-powerful and more efficient then many others.

Solution

1. Alice generates two random bit strings, R1 and R2.
2. Alice generates a message consisting of her random string and the bits (actually, maybe a few bits) that she wishes to commit. (R1, R2, b).
3. Alice calculates the one-way function value of the message and sends the result and one of the random strings to Bob. H(R1, R2, b), R1.

This transmission from Alice is evidence of commitment. In step (3), Alice uses a one-way function to prevent Bob from inverting the function and determining the bit. When it's time for Alice to show her bits, the protocol continues:

4. Alice sends the original message to Bob (R1, R2, b).
5. Bob calculates the one-way function value of the message and compares the value and R1 with the value and random string received in step (3). If matched, the bits are valid.

Advantage The advantage of this protocol is that Bob does not have to send any message. Alice sends Bob a message that promises bits, and another message that reveals the bit.

Principle The result Alice promises is a one-way function transformation of the message. Alice cannot cheat and find another message (R1, R2', b') that satisfies H(R1, R2', b') = (R1, R2, b). Alice promises the value of b by sending to Bob R1. If Alice does not keep R2 secret, then Bob can calculate H(R1, R2, b') and (R1, R2, b) and compare which one is equal to what he received from Alice.

2.2.2. Pseudo-random generator

Assumption After the two stages (commit and revealing) the protocol must obey the following: for all probabilistic polynomial time Receivers, for all polynomials p and for large enough security parameter n

1. After the commit stage Bob can not guess what he has with probability more than $1/2 + 1/p(n)$
2. Alice can reveal only one value and if she tries to cheat she is going to be caught with probability at least $1 - 1/p(n)$

Solution For a vector $\vec{R} = r_1, r_2, \ldots, r_k$ with $r_i \in \{0, 1\}$ and with exactly q indices i such that $r_i = 1$ let $G_{\vec{R}}(s)$ denote the vector $\vec{A} = a_1, a_2, \ldots, a_{1k}$ where $a_i = B_{j(i)}(s)$ and $j(i)$ is the index of the i $-$ th 1 in $\vec{R}$. If $e_1, e_2 \in \{0, 1\}^q$, then $e1 \oplus e2$ denotes the bitwise XOR of e_1 and e_2.

A. Commit Stage:

(a) Bob selects a bit string s as a random vector $\vec{R} = (r_1, r_2, \ldots, r_{2q})$ where $r_i \in \{0, 1\}$ for $1 \leq i \leq 2q$ and exactly q bits are 1 and sends it to Alice.

(b) Alice computes $c = E(b1, b2, \ldots, bm)$. Alice select a seed $s \in \{0, 1\}^n$ and sends to Bob $e = c \oplus G_{\vec{R}}(s)$ (the bitwise XOR of $G_{\vec{R}}(s)$ and c), and for each $1 \leq i \leq 2q$ such that $r_i = 0$ she sends $B_i(s)$.

B. Reveal stage: Alice sends s and $b1, b2, \ldots, bm$. Bob verifies that for all $1 \leq i \leq 2q$ such that $r_i = 0$ Alice had sent the correct $B_i(s)$, computes $c = E(b1, b2, \ldots, bm)$ and $G_{\vec{R}}(s)$ and verifies that $e = c \oplus G_{\vec{R}}(s)$

2.2.3. Complexity

Alice can reveal only one possible sequence of bits, except with probability less than 2^{-n}

For any pair of seeds $s1$ and $s2$, the probability that it fools R is at most $(1 - e/2)^q$, where the probability is taken over the choices of R. The number of bits exchanged in the protocol is $\mathcal{O}(q)$, and when amortized over m bits it is $\mathcal{O}(qm)$ which is $\mathcal{O}(1)$, since C is a good code.

For $m > n$, the communication cost is $\mathcal{O}(m)$.

2.2.4. Application

Commitments are used in zero-knowledge proofs for two main purposes: first, to allow the Prover to participate in "cut and choose" proofs where the verifier will be presented with a choice of what to learn, and the Prover will reveal only what corresponds to the verifier's choice. Commitment schemes allow the Prover to specify all the information in advance in a commitment, and only reveal what should be revealed later in the proof. Commitments are also used in zero-knowledge proofs by the verifier, who will often specify their choices ahead of time in a commitment. This allows zero-knowledge proofs to be composed in parallel without revealing additional information.

2.3. Millionaire Problem

2.3.1. Problem Description

The problem discuss two millionaires, Alice and Bob. They want to know who is much richer but both of them are not willing to let the other millionaire how rich he is. This problem is a classic secure information exchange problem which has already been solved.

2.3.2. Algorithm

Let binary string $s = s_n s_{n-1} \cdots s_1$ to represent the money. We then construct two set of string as follows: $S_s^0 = \{s_n s_{n-1} \cdots s_{i+1} 1 | s_i = 0\}$ and $S_s^1 = \{s_n s_{n-1} \cdots s_i | s_i = 1\}$ We can prove that if $a > b$, the two sets $S_a^1,\ S_b^0$ should have a common element.

2.3.3. Secure Communication

To explain the algorithm more easily, we first focus on a simpler problem. There are four kinds of salary for Alice and Bob may have: 10$, 20$, 30$, 40$. We made four lockers for them with corresponding salary tags on the lockers. First Alice put an item in each locker with a tag that has a higher salary than Alice's. Then Bob open every locker with a tag that has a lower salary than Bob's. If Bob find an item in one of the locker, then Alice has a lower salary than Bob.

We extend this small problem in this way: We replace the salary on the lockers to be S_b^0. If there is an item in one of the lockers when Alice check them with S_a^1, Alice is richer than Bob. There is a complicated algorithm to use bitwise rotation and XOR operation to provide implement which can ensure the the server cannot get any information.

3. Our Proposed Solution

Based on the the literature research we have done so far and hinted by Manuel, we have a temporary design for this system.

User Alice commits whether she's interested in the other users of this system. For instance, she's interested in user Bob and a bit 1 is committed. On the other hand, user Bob also has committed a bit for Alice, which in our case for instance is 0. After all users have finished their commitment, the server starts to do the matching. The server will inform all users to set up an secret encryption key with each other by Diffie–Hellman key exchange protocol. With this agreement, such as their public keys and secret strings, user A can insert his private bit in between two bit strings and form a message. The former bit string is the encryption of B's secret string under B's public key while the latter bit string is the encryption of A's secret string under B's public key. B can form his/her message in a similar fashion. The server will then do AND operation over their bit strings and send back the computed result to A and B. As both A and B know clearly about the length of bits they encrypted such that they can get the real result in our example 1 AND 0 easily. On the contrary, the server can only see a long string of zeros and ones and cannot get any meaningful information. Based on AND operation, user B will know nothing from the other side since B commits an 0.

Note: this solution need to be verified further.

4. Limitation

It's obvious this solution is not efficient with $\mathcal{O}(n^2)$ complexity and thus cannot scale. Also, some information is leaked because A will know B is not interested in him when he obtains a 0 but commits an 1. But this is not too unacceptable because what we care most is the case B is not interested in A but she/he finally knows that A is interested in her/him.

References

John Mikhail, Emad Farag, Edgar Minasyan, Malek Ben Romdhane, *Cryptographic Dating*.

Hristo Lulev, *Overview of Bit Commitment Schemes*.

John Ioannidis, Angelos Keromytis, Moti Yung (Eds.), *Applied Cryptography and Network Security*