Using Secure MPC to Play Games

Peinan Chen, Shruthi Narayanan, Jeffrey Shen

Table of Contents

Abstract

Yao's garbled circuits, utilized for secure multi-party computation (MPC), allows multiple parties to compute an arbitrary boolean function on their individual inputs without revealing information about those inputs, so long as parties are semi-honest [1]. While secure MPC has been examined in a number of different directions, the goal of this project is to apply Yao's protocol in a single context where it is used repeatedly. Specifically, we implemented a variant of the game Stratego using the Fairplay system [2] so that players can play without a central server and be assured that no cheating was involved. Our code can be found at https://github.com/jeffdshen/project6857.

Introduction

In one of his papers, Yao proposed an implementation of secure MPC involving a problem where two millionaires want to determine whose wealth is greater without revealing any additional information about their respective monetary values [3]. This problem, which is an example of an interaction involving secrets that must not be revealed, can be solved in a number of ways - for example, by using a variant of oblivious transfer - and therefore two people can compare secrets securely. Similarly, secure MPC can be used to play any multiplayer game in which each player has a hidden initial state and interacts with other players based on that state and all following states. Secure MPC allows the game to be played and interactions to occur without ever revealing any initial states.

The protocol used in the millionaire problem has been extended to include multiple parties for any combination of boolean operations such as AND, NOT, and OR; this extension is the one implemented by the multi-party Fairplay system [2]. In our case, we will only be involving two parties, even though generalizations with more than two players are certainly possible. For the Fairplay system with two parties, preventing one player's malicious behavior is guaranteed. The other player can be caught cheating using a cut-and-choose technique with probability $1 - 1/m$, where there are m choices [2]. Thus, we can use Fairplay to do secure function evaluation as the basis for interactions in a two-player game.

Many past studies on secure MPC [2, 4] examined using secure MPC in a one-off context. This project utilizes secure MPC multiple times, allowing previous MPC results to have an effect on future secure MPC results. Thus, in addition to assessing the feasibility of using secure MPC for a game, we also want to examine whether secure MPC functions properly when used multiple times in succession, with previous results propagating onto current game interactions and future results.

It is important to note that a lot of board games that involve secret-keeping do not actually have interactions that depend on multiple secrets that are used repeatedly. Battleship, for example, reveals the secret on each interaction, so the secret involved is not used repeatedly. In card games like Magic the Gathering, where players' hands are kept hidden from all other players, cards are fully revealed when played. We chose to implement Stratego because it involves interactions that depend on both players' secrets and neither player needs to fully reveal their secret during an interaction. Furthermore, while secure MPC would be applicable for online games like Starcraft or League of Legends, secure MPC has performance limitations that render these games infeasible to recreate [2].

Overall, the goal of this project is to use MPC repeatedly in a single context. To this end, the variant of Stratego that we will implement uses the Fairplay system for each interaction between players. We `aim to demonstrate that it is possible to create games that utilize secrets at each stage of the game without revealing those secrets.

Design

Game Specifications

In Stratego, there are two players, each with a set of 40 pieces with 12 different types of pieces. Both players play on a 10 by 10 grid and get to arrange their pieces in a 4 by 10 grid on their side of the board. Each player's arrangement of pieces is hidden from the other player. The goal of Stratego is to find and capture the enemy's Flag piece.

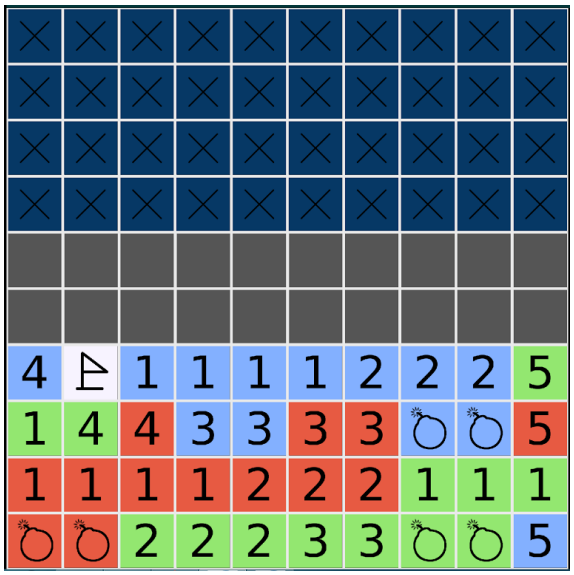

Figure 1: Example of a possible board state with our UI. Rock pieces are blue, Paper pieces are green, Scissors pieces are red, and Flags are white. Greyed out blocks are empty while dark blue, crossed out blocks are unknown, opponent pieces.

In the original variant, each player would move one piece per turn, and turns are alternated. If a player moves a piece to a square containing an enemy piece, the enemy player would be forced to reveal their piece, and one or more of the players would lose their piece. In our variant of Stratego, both players move concurrently and neither player would have to fully reveal any of their pieces.

Stratego pieces are normally assigned a value, and higher-value pieces defeat lower ones during piece interactions. In addition to this, we decided to give our pieces a rock-paper-scissors type relation. This was done to test the Fairplay system on more complicated comparisons than simple two number comparisons. Specifically, we rank pieces for each type of Rock, Paper, or Scissors from 1 to 5. Higher-ranked pieces beat lower ranks of the same type. When one type has an advantage over another type, we treat all pieces in the advantaged type as if their rank was 2 ranks higher. For example, a rank 3 Rock piece is equivalent to a rank 1 Paper piece.

Besides the rank 1-5 pieces, we also have a Bomb piece for each type. Bombs only lose to pieces that have a type advantageous to their own type. For example, a Rock Bomb loses against any Paper piece, but wins against all other types of pieces. Lastly, each player has one Flag piece that loses to everything. Flags and Bombs are stationary pieces.

Other than having different pieces and different numbers of those pieces, our game is identical to Stratego. Figure 1 shows an example of an initial board state for a player with the correct number of each type of piece.

Implementation Details

When starting a game, a player sets their initial placement of pieces. They can then choose to broadcast their game by acting as a server. Other players can connect as a client to a server player through their IP address. On each round, each player decides on a move, consisting of a starting coordinate and a direction. If either of the moves causes any piece interactions, we run Fairplay on those moves to determine which pieces win. If a Flag piece is ever taken, we end the game.

A key idea in our design is that players can commit to actions by using SHA-256 to send their hashed action. These actions can then be revealed and verified at later points in the game. Specifically, we concatenate a string representation of the following: the action being hashed, a fixed length nonce, and an identifier for whether the player performing the action is the server or the client. When a player reveals their action, they also reveal the corresponding nonce and identifier.

Therefore, as soon as the players have connected, they send a commitment of their initial board state to each other. For every following move, they send a commitment of that move to their opponent. Once both players have sent their hashes, they send their actual moves to each other. If the moves match the commitments, the game continues. Otherwise, the game is prematurely terminated.

When the game ends, each player sends their initial board state to each other. We then verify that the initial board states match the commitments sent at the beginning of the game and that all of the moves and results are consistent with the initial board states. This is done by replaying the game on both players' sides, directly comparing pieces rather than using Fairplay on conflicts. If any verification fails, we report this to the users. Otherwise, we output the result of the game and that the result was verified.

Analysis

Security Correctness

The security of our implementation depends on Fairplay's security and the security of our commitments. Fairplay utilizes Yao's garbled circuit protocol, which is secure against semi-honest adversaries. Specifically, it is secure when the adversary runs the programs and algorithms correctly, but might look at information passed between actors [1]. The basic idea of Yao's garbled circuits is that a peer, P1, transforms a circuit by changing each gate's computation table to correspond to new random keys. P1 then sends the circuit to P2 together with the keys corresponding to P1's inputs. Since P1 has the mappings from his inputs to keys, even if P2 gets P1's keys, P2 will not be able to determine those keys' values. P2 uses oblivious transfers to receive from P1 the keys corresponding to P2's inputs and then calculates the output to the circuit and outputs the result.

Oblivious transfers are a type of protocol in which a sender transfers one of potentially many pieces of information to a receiver, but remains oblivious as to what piece (if any) has been transferred [2]. For example, P2 could ask for the keys for both 1 and 0 as inputs to a gate from P1 without revealing whether he has 1 or 0 as his secret. By using oblivious transfers, P2 can acquire all the keys for his inputs without revealing those inputs to P1. Thus, neither P1 nor P2 knows each others' secret.

Yao's garbled circuit protocol is secure against malicious P2 adversaries but not against malicious P1 adversaries since P1 can create an incorrect circuit. However, Fairplay has measures in place to thwart malicious P1 adversaries [2]. Instead of sending just one circuit to P2, P1 sends m garbled circuits to P2. Then, P2 randomly chooses a circuit that will be evaluated and P1 reveals the secrets associated with the remaining circuits for evaluation. P2 then verifies

that these $m - 1$ circuits match the pre-garbled circuit. Lastly, both P1 and P2 evaluate the chosen circuit. This method catches a malicious P1 adversary with probability $1 - 1/m$.

We discovered that our commitment scheme is not necessarily provably secure, as mentioned in Section 1.3 of a paper by Halevi and Micali [6]. Our commitment scheme is resistant to a number of common attacks such as brute force, replay, and length extension attack. The nonce prevents brute force attacks by increasing the input space, and the identifier prevents replay and length extension attacks since now no extension of the string to be hashed forms a valid action for the other player. There may be probabilistic attacks in which an adversary is able to distinguish between two candidate moves. However, Halevi and Micali do, however, provide a relatively simple, practical commitment scheme that is provably secure and which can easily replace our existing scheme.

Overall, Fairplay, and thus our implementation, is completely secure against semi-honest adversaries and secure against malicious adversaries with probability $1 - 1/m$, given a provably secure commitment scheme.

Runtime

One concern that the Fairplay paper mentions is that the protocol is not very practical since computations are costly [2]. However, we did not experience any significant performance limitations. Specifically, during our testing, our compiled Fairplay circuit consisting of 162 gates completed under a second while on LAN. However, since each round of Fairplay requires several oblivious transfers, computation time is limited by network speeds and by the size of the circuit. Thus, Fairplay might not be practical for other online games that require many more network interactions and larger circuits to process more inputs, such as League of Legends.

Conclusion

We implemented a proof of concept for a game that has multiple interactions based on player secrets without relying on a central server. The game uses the Fairplay system to complete interactions based on secrets without revealing the secrets. Our implementation can be secure against malicious actors with high probability and does not have noticeable delays. The ideas that we use here are fairly general and can be applied for other games as well. In the future, we could work on trying to improve the runtime of our system since there exists faster implementations of garbled circuits [5]. We could also consider improving our commitment scheme [6] as well as extending our system to multiplayer games.

References

[1] M. Bellare, V. T. Hoang, and P. Rogaway, Foundations of garbled circuits, *Proceedings of the 2012 ACM conference on Computer and communications security*, 784–796 (2012).

[2] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella, Fairplay - A Secure Two-Party Computation System, *Proceedings of the 13th conference on USENIX Security Symposium*, 20-20 (2004).

[3] A.C. Yao, Protocols for Secure Computation, *54th Annual Symposium on Foundations of Computer Science*, 160-164 (1982).

[4] A.C. Yao, How to generate and exchange secrets, *27th Annual Symposium on Foundations of Computer Science*, 162-167 (1986).

[5] Y. Huang, D. Evans, J. Katz, and L. Malka, Faster Secure Two-Party Computation Using Garbled Circuits, *Proceedings of the 20th USENIX conference on Security*, 35-35 (2011).

[6] S. Halevi and S. Micali, Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing, *Lecture Notes in Computer Science Volume 1109*, 201-215 (1996).