Distributed Cryptographic Mafia

Ashley Wang, Cristhian Ulloa, Ronald Gil

May 11, 2016

Abstract

Mafia is a popular party game that involves drawing cards from shuffled decks to assign roles, voting via secret ballots, and communicating in secret. In all previous iterations of this multiplayer role playing game, these actions were possible due to the presence of a narrator, who acts as a trusted third party and is responsible for running the game. This paper removes the reliance on a trusted third party, and discusses several protocols that can be used to play the game in a computationally secure, distributed setting.

1 Introduction

Distributed systems pose several interesting challenges to security due to their severe limitations. For example, the computers in the system have their own independent memory and clock. This necessitates protocols that are conscious about concurrency and event ordering. However, distributed systems are also far more transparent, scalable, and potentially resistant to failure compared to a centralized server. These properties are highly valuable for world-class software systems such as MapReduce[1], the Google File System (GFS)[2], and Presto[3], Facebook's distributed SQL query engine for big data. This is an area of research, along with cloud computing, that is quickly emerging as the future of large-scale software systems.

Playing a distributed game of Mafia[2] becomes advantageous under the following limitations: that the game cannot be played physically, and there is no trusted third party to run the game. These limitations can be found in the real world in a large variety of situations. For example, information sharing between intelligence agencies in the United States must be done in a distributed manner. Allowing a central authority global access to information from the CIA, NSA, and FBI could be a catastrophic security hole, an a highly prized target for attackers. [Another example would be data sharing between hospitals or lawyers who must comply with client confidentiality standards.]

Mafia (Section 1.1) is an especially interesting case study for secure distributed systems due to the importance of keeping secrets in the game. The most important secret is the assignment of roles to players in the game. There are roles that are "village-aligned" and "mafia-aligned", resulting in two teams of players who act as each others' adversaries. In a centralized game of Mafia the neutral third party assigns these roles, but in a distributed system no one may be trusted with this information because each player would seek to use it to their advantage. A key element of the game is to create an

[1] http://research.google.com/archive/mapreduce.html
[2] http://research.google.com/archive/gfs.html
[3] https://prestodb.io

atmosphere of paranoia by keeping true identities ambiguous, and our protocol aims to maintain this environment. Other secrets that must be protected are the secret ballots made by the mafia-aligned team, and communications between mafia-aligned players. Since the functionality of this game depends on keeping these secrets, we believe that it will be a worthwhile challenge to design cryptographic protection for these secrets in a distributed system with no trusted third party.

This paper describes our design and the expected final state of our Python implementation for a distributed, computationally secure game of Mafia. Our final design leverages cryptographic schemes such as Secure Multi-Party Computations, Shamir's Secret Sharing Schemes, Pedersen's commitment scheme, and 1 out of n oblivious transfers. The protocol removes any advantage that players may gain through collusion since it is capable of detecting cheating. The protocol guarantees the security of the secrets described above, ensuring a fair game where no one side has an unfair advantage.

1.1 Mafia

There are three main stages in this game. In the setup stage, each player is assigned an unknown role. The main roles include: mafia, townsperson, doctor, and detective. In the day round, everyone votes on a player to lynch. In the night round, mafia members choose a player to kill, and the doctor chooses a player to save.

Mafia's gameplay can be described as follows:

1. Assign roles to players without other players discovering the roles of other players.
2. Allow all the players to communicate with each other during the 'day' to discuss who to lynch.
3. Remove the lynched player.
4. Allow the mafia members to communicate with each other in order to pick who they want to kill.
5. Allow the mafia, doctor, detective, and vigilante to make their choices in secret.
6. End the round, or the 'night' in the game once all of the special roles above have made their choice.
7. Once the round has ended, reveal the 'results' of the round such as who has died, who might have been saved, etc.
8. Remove the 'dead' player.
9. Repeat steps 2-8 until a endgame condition has been met. The two possible endgame conditions are the deaths of all the mafia member of the deaths of all the villagers.

1.2 Previous Work

To the extent of our knowledge, there is no existing literature on creating a distributed cryptographic version of Mafia. However, there have been several case studies of how games would operate in a distributed setting. Due to the large number of common elements between games, we found these papers highly useful in designing our protocols.

One of the earliest papers we found in the area of distributed gaming is "Mental Poker" [9]. A key insight we gained from this paper was the concept of a "fair deal" from a card deck. In Mafia, a "fair deal" is analogous to a fair distribution of role assignments. This is the foundation for starting a game in which no player has an unfair advantage.

Several previous 6.857 projects, such as "Distributed Game of Settlers of Catan"[4] and "ROCK, PAPER, SCISSORS...Cheat"[7] also presented designs for distributed games. Although there were no protocols that were directly applicable from one game to another, we were able to borrow several key concepts in order to design our own protocols. Many aspects of our threat model such as the honest-but-curious adversary model and cheater detection failsafe come from these papers. Several other strategies that were common across all these distributed gaming papers were the Pedersen's Commitment Scheme, Secure Multi-Party Computation, and 1-out-of-n oblivious transfers.

2 Threat Model

Our system is set up as a distributed network of players. If there are n players, then we will have n computers in the network. No single server s_i should have access to all of the secrets in the game. In other words, there is no central server (besides a public chat server independent from the rest of the game). The reason we have constructed the system this way is twofold. First, we must trust that this central server is honest and does not share information that can provide another player an unfair advantage. Second, a central server, while honest, may become a single point of failure. If the attacker gains access to the central server, then they may gain the upper hand during the game.

We assume players have polynomial time constraints for their attack vectors. Given this constraint, we assume players will try to use any information they can collect in order to decipher the identity of other players as well as influence the results of the game. This makes them reasonably active in the system, but rules out active attacks such as a man-in-the-middle attack. For example, we assume that all players are able to detect traffic between servers in the game and will try to decrypt any messages that are sent. We may also assume that players are able to analyze network traffic to guess the identities of mafia members.

We assume the players will still want to play a functional game. This precludes actions whose sole result would be to stop the game from progressing further or concluding, such as DDOSing.

We mitigate bad behavior by detecting cheating in the game and potentially identifying the culprit. By implementing measures to detect cheating in the game, we assume that there will be social repercussions to being caught and leave it to the players to determine the punishment.

We also assume that no player wants to sabotage their own team. For example, a mafia member that deliberately leaks information about other mafia members is not within our threat model. However, we do address the possibility of multiple players on the same team working together to attempt to cheat the system.

Lastly, the overarching goal is to prevent any extra advantages from actions that would not already exist in the real game. For example, in the real life game, players

who know each others roles because they are colluding outside of the game would still have that advantage in this version but should not gain any extra information.

3 Protocols

In the following protocols, we assume some random ordering of the players has been agreed upon where the first player in that ordering is referred to as Player A and the last is referred to as Player N. Whether the ordering is truly random or not should not impact the security of the game.

3.1 Setup protocol[4]

The purpose of this protocol is to assign the roles of mafia, doctor, detective, and townsperson to each player. The assignment of these roles is the most important secret in the game, and should be protected with computationally secure encryption.

Our protocol is as follows:

- Stage 1: Player A creates a card for each role. We define a card to be of the form (x_i, c) where x_i is a publicly known constant that corresponds to a role (e.g. the name of the role) and c is a boolean that is True if the card has been claimed.

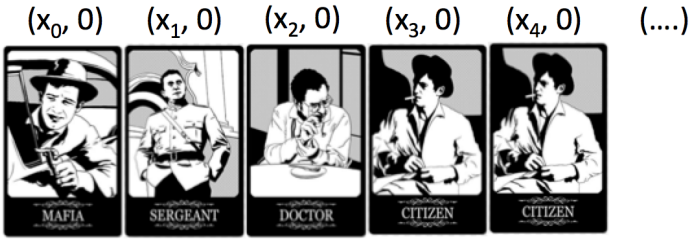

Figure 1: an example 'card deck' or list of roles created by Player A at the beginning. The tuple (x_i, c) is the Python representation of this card, where 0 means unclaimed and 1 means claimed.

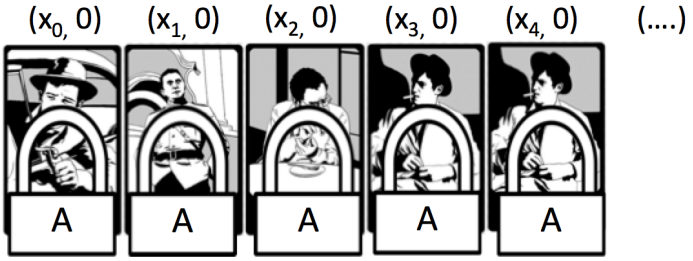

Figure 2: the lock icons on each of these cards represent an encryption using Player A's secret key.

[4]Construction inspired by [9].

Player A encrypts[5] the x_i value of the card. Then Player A scrambles the list of all cards and sends it to Player B.

- Stage 2: Player B will randomly choose a single unclaimed card, encrypt the x_i value, and scramble the cards. Then Player B sends the cards to Player C. This process is repeated by every player until it reaches Player N whom sends the cards back to Player A.

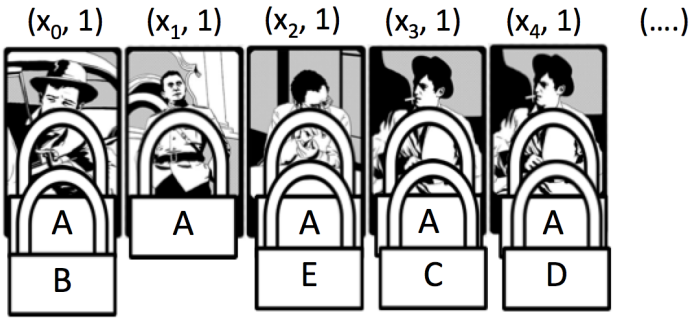

Figure 3: the state of the card deck at the end of this stage will look like this.

- Stage 3: Player A decrypts the x_i value of every card and keeps the x_i value that produces valid plaintext as his or her role. Then Player A removes the card with the x_i belonging to him or her, keeps it, and sends the remaining cards to Player B. This process repeats until the process reaches Player A again. When the card deck, there should be no cards left.

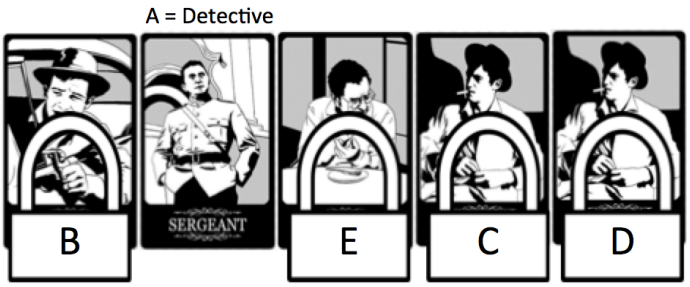

Figure 4: Once all the A locks have been removed, A can see his or her own role, but does not know any information about the roles of other players.

Through this protocol, we successfully prevent all players, including Player A, from determining which players received which roles. Furthermore, even if Player A were to try to specify an order to the cards, the random shuffling prevents Player A from influencing the role assignment process.

[5]This and all later encryption/decryption for this protocol assumes a commutative encryption function.

3.2 Voting protocols

3.2.1 Day Round

The Day Round is a seemingly straightforward part of the Mafia game, where the goal is for the village of players to lynch a single player. In a distributed system, however, we need to ensure the following:

1. No player can change another player's vote.
2. No player can vote twice or have their vote counted twice.
3. Dead players cannot vote.
4. All players must vote before lynching.
5. In the event of a tie, everyone revotes.

The Day Round is implemented as a Pedersen Commitment Scheme, where instead of committing to a bidding price each player commits to a vote. This prevents players from changing their vote after seeing the final results.

- Stage 1: Each player votes for the player they would like to lynch, and creates a commit $c = g^x h^r$ where g is a generator and $h = g^a$ where a is a secret r is a random number and x is the vote.
- Stage 2: Each player reveals their commit, and records a list of everyone else's commits.
- Stage 3: Each player reveals their vote and their proof, a tuple (x, r). Each player uses this information to verify the commits and compiles a voteList.
- Stage 4: If the vote list results in a tie, the procedure starts over and everyone revotes.

Pedersen's Commitment Scheme provides information-theoretically secure hiding and is computationally binding. Each player maintains a list of alive players, and only looks at the commits and votes of these players to ensure that dead players cannot influence the game. In addition, each individual player is responsible for displaying their commit and vote information publicly and tallying the overall score themselves. With this strategy, each player comes to their own conclusion about which player has been lynched. Each player should reach the same conclusion about who has been lynched; otherwise, it is assumed something went wrong, e.g. someone cheated, and the game is ended. The main purpose in using the commitment scheme was to prevent any players from changing their vote after finding out how other players have voted.

3.2.2 Night Round

Unlike the day round, the night round requires that players communicate with each other while keeping their identity a secret. Specifically, when the members of mafia vote, no one outside of the mafia should be able to identify them as a member of the mafia. The same is true for the doctor. This imposes several restrictions to our protocol. For example, unlike the day round, the mafia members cannot publicly share their vote. To address these requirements, we created the following protocol:

- Stage 1: First, the mafia vote occurs. This requires a secure multiparty computation to execute addition among the alive players.

- Stage 2: Each mafia member should input the player they have decided to kill, and everyone else should input a zero.
- Stage 3: To get the player who was killed, the output of the computed function (addition) is divided by the number of mafia members.
- Stage 4: If this produces a jumbled value, the process is restarted. If any mafia member disagrees with the final value, the game is ended since this requires the person to reveal their role.
- Stage 5: This same process is repeated for the doctor.
- Stage 6: If the person to be killed and the person saved are the same, nothing happens (no one dies). Otherwise, that person is considered to be killed from then on.

3.3 Mafia Private Communication (MPC) protocol[6]

Before the mafia votes, they have to agree among themselves what player they want to kill. This requires the members to communicate secretly, where other players cannot read their messages nor identify who the members of the mafia are. To achieve these goals, we created the Mafia Private Communication protocol outlined below.

- Stage 1: Player A creates a safe prime N and a generator g for that prime.
- Stage 2: Player A includes g and N with the mafia roles (in the plaintext) before sending them out.
- Stage 3: When everyone has their roles, those who are mafia create a secret m_i and compute g^{m_i}, while everyone else makes some random number. Then every player broadcasts either g^{m_i} or the random value to every other player.
- Stage 4: Each mafia member then raises the number they get from each other person, call it g^{m_j}, to their secret m_i in order to make the key $g^{m_i \cdot m_j}$. They then reply with a test message encrypted with that key.
- Stage 5: Only people that transmitted actual g^{m_i} values should have been able to decrypt and send back proof that they read the message.
- Stage 6: Mafia members can then communicate with each other to create a random shared symmetric key that all of them know, allowing them to communicate securely with each other.
- Stage 7: Each mafia member can verify at the end of the round if he/she is communicating with the right number of mafia members based on the votes in the published round results. If a mafia member is communicating with the incorrect number of members, cheating is assumed and the game is ended.

To keep mafia members' identities secret, mafia members communicate by broadcasting a message encrypted with the new secret key while everyone else sends a random value. Only mafia members who have the key can decrypt the actual messages. Since everyone sends a seemingly random message, the identities of mafia members remain hidden.

[6] A large part of this process is simply Diffie Hellman key exchange[6].

3.4 Detective Verification

In the game of Mafia, the detective is allowed to query the game moderator to determine if a specific player is a member of the mafia or not. There are a few guarantees included during this process, namely:

- The identity of the detective remains unknown to all other players.
- The identity of the player the detective was questioning is not revealed. This also means that the player in question does not learn that the detective has been investigating him/her.
- If the player in question is not mafia, no other information about that player's role is revealed. The detective only learns that the player is not a member of the mafia.

In our scheme so far, the information about a player's role is a secret that only the players themselves know. Since we do not want the player to know if the detective is investigating them, we decided to use an oblivious transfer scheme.

Initially we considered having Player A, the same player that creates the roles, create the secret (the role) and break it into multiple pieces that would be shared with the n players. However, this would require player A to know the role that corresponds to a player, which breaks the game requirements.

We later considered having each player create their own secret that they would break into shards and share all but one shard with all other players. This way no single player can determine what the role of a player is. The detective then requests each player to give the corresponding shard when querying about a specific player. This scheme, however, has two flaws. First, it reveals to every player the player that is being investigated and also allows honest but curious adversaries to determine who the detective is.

To keep the detective's identity a secret, we considered using Secure Multi-Party Computation to allow all users to input data but only have the detective's query be output. Then, one of the n players, say player A, would take the output and retrieve the shard that it has that is associated with the specified output. It then passes it on to the next player who also retrieves its respective shard, combines the pieces, and passes it on. This repeats until it reaches player A again, at which point player A publishes the result. However, to prevent every player from reading the result, the detective, during the secret building process, does not add the shard that it holds but instead adds a random value. Therefore, the result published by player A does not actually reveal whether the player in question is mafia or not. Given the result published by player A, the detective can undo the operation with his random value that he/she had previously added and replace it with the true shard. This then gives them the information they requested while maintaining the the detective's identity secret. However, the identity of the player being queried is still known.

With the previously mentioned schemes, if any player dies, they must broadcast the list of mappings from player to shard that they own so that the information is not lost. Furthermore, since the other shards are still privately held by the live players, the shards do not reveal the identities of the remaining players. If the shard that is revealed is the last shard that needed to be revealed for a player, the shard will belong to the player that just died at which point his/her identity will be revealed anyway, so no new information is gained.

A problem with having players create their own secret and break it into shards is that there is no way to make sure that they will not lie about their identity. To address this we have every player individually keep track of all the secrets revealed as players die. When the game is about to end, all remaining players must publish all shards they hold and will build all the secret keys given all the pieces that were published. If the identities provided by the secrets do not match the roles created during the setup process, we know someone has lied about their identity and we declare the game invalid.

4 Implementation

Our implementation architecture is as follows: each player is a server that listens to one port in order to communicate with other players. Communication between players happens via HTTP POST and GET requests. The system assumes that players are connected to each other via the internet and that the connection is reliable: at least with the guarantees provided by TCP and with disconnect times no longer than around half a minute.

For this project we successfully implemented some of the previously mentioned protocols. Our implementation was done using the python libraries tornado[7] and cryptography[8]. We also used available implementations of the Diffie-Hellman[9] and Milller-Rabin[10] algorithms which we modified to meet our requirements.

We have successfully created the setup protocol described earlier. This includes the information necessary for creating a secure private communication mechanism for members of the mafia. With our implementation, players can also cast their votes during the day round.

We were unable to implement the night round due to the lack of available libraries implementing secure multi-party protocols. We were, however, able to come up with a possible circuit in the vein of Yao's garbled circuit in the event of future implementation.

Our code can be found at `https://github.com/ashleywang1/857MafiaGame.git`.

5 Challenges

For the project we came across a couple of challenges. First of all, during the design phase we came up with a couple of schemes to give the detective the ability to determine if a player is a member of the mafia. Unfortunately, as we mentioned before, none of these schemes meet all our requirements. Giving the detective this ability was the last major design we needed to make in order to complete the system.

During implementation we also came across a couple of challenges. One such issue arose because ports are not able to handle more than one request at a time. This frequently led to concurrency issues that required a lot of time to debug. Another issue we encountered was the need for verification. We need a mechanism that ensures that we are always getting information from the correct source. Unfortunately, signatures need a trusted third party, which we are not allowed to have. However, if we assume a

[7] http://www.tornadoweb.org/en/stable/
[8] https://cryptography.io/en/latest/
[9] https://github.com/lowazo/pyDHE
[10] https://gist.github.com/Ayrx/5884790

curious but honest adversarial model, we can query each port separately to ensure that we get the information straight from the source.

5.1 Future Work

Work that still needs to be done for the project is the implementation of the night round, specifically voting at night as well as the protocol for detectives to investigate players. Once these protocols are implemented, all that is left to do is join these protocols along with the mafia communication scheme.

We would like to use the Viff library to implement an actual garbled circuit for use with the Secure Multi-Party Computation for the night round and verify that it is computationally secure.

Our current design allows the use of a central server, external to our system, that is used a the messaging service for players during the day round. This is allowed in our protocol since the server is not trusted but also does not have the ability to affect game play.

There also exist many variations of the mafia game. Specifically, some variations have more roles. We believe that extending this game to include a variety of more complex roles would be an interesting challenge and a useful feature to include in the game.

6 Conclusion

In this paper, we posed the question of whether or not it is possible to replace a trusted third party in a relatively complex game with secure cryptographic protocols. To answer this question, we designed a proof-of-concept that uses several computationally secure cryptographic schemes to replace a central server. We took this further by implementing some of the schemes, including a computationally secure round-robin, multi-party card dealing scheme to assign player roles.

Our research revealed a few options for protocols that may be used for the portions of the game that have not yet been implemented. With more time and research, we believe that these preliminary schemes could be modified to provide the secure functionality required.

References

[1] Shamir, A., Rivest, R.L., Adleman, L.M. 1981. Mental Poker. The Mathematical Gardner:37-43.

[2] Mafia rules: https://en.wikipedia.org/wiki/Mafia_(party_game)#Gameplay

[3] Shamir's secret sharing: https://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing

[4] Previous 6.857 distributed game of Settlers of Catan: http://courses.csail.mit.edu/6.857/2014/files/07-alsibyani-mickel-vasquez-zhang-distributed-settlers-of-catan.pdf

[5] Socialist Millionares procedure that we plan to use: https://en.wikipedia.org/wiki/Socialist_millionaires

[6] Diffie-Helman procedure that we plan to use: https://www-ee.stanford.edu/~hellman/publications/24.pdf

[7] Previous 6.857 Rock Papers Scissors game in a distributed network: http://courses.csail.mit.edu/6.857/2015/files/chen-hamlin-lim-muco.pdf

[8] Previous 6.857 Secure MultiParty Encryption http://courses.csail.mit.edu/6.857/2016/files/chen-narayanan-shen.pdf

[9] Poker game in a distributed network: Shamir, A., Rivest, R.L., Adleman, L.M. 1981. Mental Poker. The Mathematical Gardner:37-43.

[10] Secure Multiparty Computation http://repository.cmu.edu/cgi/viewcontent.cgi?article=1004&context=jpc

[11] Yao's Garbled Circuits https://www.cs.uic.edu/pub/Bits/PeterSnyder/Peter_Snyder_-_Garbled_Circuits_WCP_2_column.pdf