MIECT: Security 2019-20

Secure, multi-player online card game

Presentation of final project: 1st and 2nd weeks of Jan, 2019
Deadlines: Dec 31 (final project)

André Zúquete

Changelog

- v1.1 - Note regarding the use of the system using only one Citizen Card.
- v1.0 - Initial Version.

1 Project Description

The objective of this project is to develop a system enabling users to create and participate in online games (for simplicity, we will assume only card games). The system is composed by a croupier (or table manager) and a bounded set of players. The system should be designed to support the following security features:

- **Identity and authentication of players**: All players must identify and authenticate to each other using strong identification methods. Namely, they should authenticate to each other using a Citizen Card. The croupier will only start the game after getting a signed statement from all players including the identity of the opponents. This is fundamental to ensure that no player is playing with someone that they do not want to play with;
- **Confidential and random card distribution**: the croupier creates the card deck and initiates its distribution to the players. The distribution should involve all players, and all of them should contribute to the randomness of the card distribution and to the guarantee that no single entity is able to know, for sure or with a high success probability, the cards of any other player.
- **Honesty assurance**: players must play with the cards they have re-

ceived (their hand), and cannot use any other. To enforce this, players must commit to their hand before the start of the game. The commitment must not reveal the cards in the hand and must be properly signed.

- **Correct game evolution**: the game evolution should be orchestrated by the croupier. On the other hand, the croupier and the players can collaborate to catch cheaters. For instance, the croupier and the rest of the players can detect a card played twice, or when a card was played in a situation where it could not have been used. And players can detect if another player is using one of their cards. In the first situation, either a wrong play can be denied a anteriori, and the game can proceed upon a correct play, or a wrong play can only be detected a posteriori, and the game needs to be aborted. In the second case, all players must show their initial hand by revealing the data that originated the bit commitment; the cheater should not be able to do so.
- **Correct accounting**: at the end of each game some accounting must be done relatively to the outcome of the game (e.g. money to pay or receive). Such accounting must be agreed by all players, and they should keep to themselves such agreement in order to properly handle future disputes.

2 Project Description

2.1 System Components

We can consider the existence of two main components: (i) multiple clients, through which players interact, and (ii) one server, which serves as croupier for all clients to connect. A croupier can serve several games simultaneously.

Ultimately, each player can develop their own client application, but this is not the goal of this project. Therefore, each group will develop a single client, which can be run by 4 different players (in different hosts, if necessary).

2.1.1 Croupier (server)

This server will expose a connection endpoint through which clients can exchange structured requests/responses with it.

The croupier is the system component that forms a table of players, creates the deck, initiates the deck distribution (with the help of the players), controls as much as possible the sequence of the game, receives the complaints of the players, makes the game accounting and receives accounting confirmation from players.

The croupier is also the rendez-vous point for players. Before being able to play, players should present themselves to the croupier (show their identity) and prove their identity ownership. Then, a player can browse through all available layers and choose a required amount of players to forma a table.

Random table setups are also a possibility, but not under the control of the croupier. Players should state whether or not they are available to be randomly selected by other players to join their table, and the formation of a table follows the previous patterns, but using randomness to select partners instead of pinpointed people.

2.1.2 Player (client)

A player is an application that interacts with a user, enabling they to create and participate in games. This application needs to interact with the user Citizen Card for producing digital signatures in order to:

- Prove their identity (to the croupier and to other players);
- Commit to a given hand;
- Play a given card;
- Protesting against an illegal play; and
- Agree with an accounting upon a game.

Throughout the interaction between a player and a croupier, or even pairs of players (see below), they both need to prove to themselves that they are authentic, which requires authentication, but not with a strong mechanism such as Citizen Card's signatures. For this, they may use the concept of session and use session keys.

Suggestion: during the registration process clients may provide a value that enables them to establish different session keys with the croupier and all other players without having to use the Citizen Card again for that purpose.

2.2 Processes

There are several critical processes that must be supported. Students are free to add other processes as deemed required.

- Join/leave a croupier;
- Create/join a game table;
- Commit to a card hand;
- Play cards from the given hand;
- Check the cards played by others;

- Complain against cheaters;
- Accept the outcome of a game;
- Agree with accounting decisions; and
- Store accounting receipts.

Several types of card games are possible, but only one needs to be implemented. Choose a simple game (e.g. Hearts[1]), games with trump suits are harder to deal with (you need to ensure an honest trump suit selection).

It is strongly suggested to structure all exchanged messages as JSON objects or Google Protocol Buffers. JSON is a very user-friendly textual format and there are many libraries for building and analysing JSON objects. Binary content can be added to JSON objects by converting them to a textual format, such as Base-64. Google's Protocol Buffers[2] has the advantage of creating functions to perform the marshaling and unmarshaling of data into and from exchanged messages.

2.3 Security assumptions

All entities can cheat; no individual honesty is assured. But, if possible, the system should be immune to collusion. Namely, no two entities, either two players, a player and the croupier, or two players and the croupier, can derive with complete precision the hands of the other players. It is trivial to show that three colluded players can derive the game of the third player, and there is no technology that can solve this problem other than to prevent the possibility of collusion.

3 Cryptographic protocols

A bit commitment is a value that stands as a commitment made by someone relatively to some action performed (or data created) in the past that can be shown in the future to hold. In the meantime, the bit commitment does not allow others to tell which actual action (or data) were committed by its creator. In this project, a bit commitment of a hand is a computation performed over the set of card that form a hand which can be revealed before the beginning of the game without revealing the cards. At the end of the game, a player can demonstrate their honesty by showing the data that was used to compute their bit commitment, thus enabling others to check whether or not they played with the right cards.

[1] `https://en.wikipedia.org/wiki/Hearts_(card_game)`
[2] `https://en.wikipedia.org/wiki/Protocol_Buffers`

Bit commitments can be computed in many ways. One of them consists in using a one-way hash function h and two random values. Assuming that the value to commit with is C (cards), the bit commitment b can be computed as

$$b = h\left(R_1, R_2, C\right)$$

and published as (R_1, b). To prove the correctness of this bit commitment one has to publish (R_2, C).

The deck randomization and hidden selection of cards per player is not trivial, but there are some solutions. One is the following, that runs in four stages:

1. Randomization stage: the deck of N cards is created as a set of N elements with the value and suite of the card (e.g. queen of spades). The croupier and each player receive the deck at the time, from the previous one, shuffle the cards randomly and apply an encryption per card (the same key, per participant, must be used for all cards).

2. Selection stage: the encrypted deck circulates among players. Each player receives the deck and randomly selects one of two action: pick up a (random) card or back off.

   In the first case, it removes the (encrypted) information from the deck.

   In the second case, two options can happen: maintain the deck as it is or swap part (or the whole) of the cards already picked up by itself by other cards from the deck.

   At the end, the deck is sent (directly) to another player, randomly selected. The probability of card pick-up should be low (say, 5%), in order to increase the difficulty of tracing the effective distribution protocol.

   When the circulating deck is empty, any player can randomly activate the two final stages, which can follow a fixed order among players.

3. Commitment stage: any player can start it, randomly, when the deck is empty. The player replaces the deck information by a bit commitment vector (one entry per player), adds it signed bit commitment of the received hand (still encrypted) and forwards it to the next player for continuing the collection of bit commitments.

4. Revelation stage: a player, after getting the bit commitment of all the other players, adds to the circulating information its encryption key and forwards it to the next player.

   Any player, after receiving a message from another player that does not include new information, can terminate this stage and inform the

croupier that the deck was distributed and the game can start.

Note that only at this stage players know the cards that belong their hand but cannot change them, because of their bit commitment.

The communication between players must be confidential, to prevent third parties from eavesdropping it. Such communication can pass through the croupier, for making it easier to implement the player-to-player communication, but the croupier must not understand the data exchanged. Therefore, players must establish a session among them, one per each pair of players, with a temporary session key for protecting the communication. Session keys can be used to ensure confidentiality or authenticity.

Note, however, that the authenticity of actions or data that must be checked by all other players, such as the bit commitments or each card played, must be ensured with digital signatures performed with the Citizen Card.

In protocols that take a variable number of iterations to increase confusion, it is often crucial to maintain the size of the exchanged messages in order to prevent eavesdroppers from performing side-channel attacks, namely to learn how the protocol is evolving over time. Thus, all messages exchanged during the deck distribution protocol should be of equal size, regardless of the number of cards already pick up so far by the players.

4 Suggestions

The communication between players and the croupier, or with other players, is easier to implement with TCP streams, one per player with the croupier. Direct communication between players can be routed through the croupier.

Most of the time the action of the players consists on receiving orders (or data) from the croupier, or from other players, and to respond accordingly. For instance, agree to join table, perform the deck distribution, play a card, accept the outcome of a game. The main exception is when a user decides to create a table, in which case its player leads a table selection process before performing the actions previously referred. Consequently, one implementation alternative that can be considered is to create two client applications:

- one to manage tables; and
- another one to play on a given table.

The croupier server can use a thread for handling table management and a thread for handling each table. Since each table's thread may need to receive information from any of a given set of players, table handlers may benefit from the `select` Linux system call (there is a similar one in Windows), which is available with the same name in the Python's standard library.

For Java, however, the best is to use a thread per player socket to pick up messages and add them to a unique, synchronized queue red by the table manager thread. Please, do not use message polling!

You can use the croupier server to distribute certification chains to players, in order to enable them to validate the signatures provided by other players.

In order to enable the system to be tested with 4 players and only one Citizen Card (a genuine or fake one), the applications can recognise two types of signatures: ones provided with a Citizen Card, other provided without it.

5 Functionalities to implement

The following functionalities, and their grading, are to be implemented:

- (2 points) Protection (encryption, authentication, etc.) of the messages exchanged;
- (2 points) Identification of users in a croupier with their Citizen Card;
- (1 points) Set up of sessions between players and a croupier (using the Citizen Card);
- (1 points) Set up of sessions between players (using the Citizen Card);
- (1 points) Agreement to join a table (using the Citizen Card);
- (3 points) Deck secure distribution protocol;
- (1 points) Validation of the cards played during a game by each player;
- (2 points) Protest against cheating;
- (1 points) Possibility of cheating;
- (2 points) Game accounting agreements (using the Citizen Card) and storage of accounting receipts;

To simplify the implementation, you may:

- Assume the use of well-established, fixed cryptographic algorithms. In other words, for each cryptographic transformation you do not need to describe it (i.e., what you have used) in the data exchanged (encrypted messages, receipts, etc.). **A bonus of 2 points** may be given if the complete system is able to use alternative algorithms.
- It can be assumed that each server has a non-certified, asymmetric key pair (Diffie-Hellman, RSA, Elliptic Curve, etc.) with a well-know public component.

Grading will also take into consideration the elegance of both the design and actual implementation.

Up to 2 (two) bonus points will be awarded if the solution correctly implements interesting security features not referred above. But, please, implement the features required first!

A report should be produced addressing:

- the studies performed, the alternatives considered and the decisions taken;
- the functionalities implemented; and
- all known problems and deficiencies.

Grading will be focused in the actual solutions, with a strong focus in the text presented in the report (4 points), and not only on the code produced! It is strongly recommended that this report clearly describes the solution proposed for each functionality. Do not forget to describe the protocols used and the structure of each different message.

Using materials, code snippets, or any other content from external sources without proper reference (e.g. Wikipedia, colleagues, StackOverflow), will imply that the entire project will not be considered for grading or will be strongly penalized. External components or text where there is a proper reference will not be considered for grading, but will still allow the remaining project to be graded.

The detection of a fraud in the development of a project (e.g. steal code from a colleague, get help from an external person to write the code, or any other action taken for having the project developed by other than the responsible students) will lead to a grade of 0 (zero) and a communication of the event to the University academic services.

6 Project phases

The security features should be fully specified prior to start its implementation.

We recommend the following steps for a successful project development:

- Develop a complete, non-secure client and server applications. This step can start immediately.
- Produce a draft report of the security specification.
- Add secure sessions between players and a croupier and between players.

- Involve the Citizen Card in all the required steps, including the validation of certification chains.
- Add security to the deck distribution protocol.
- Add the validation of the played cards.
- Add the protest against cheating.
- Add cheating support.
- Add security to accounting and to the production of receipts.

There will be a **project milestone in November, 16 and 17**. In this milestone students should get feedback about a **written overview** (through their Code@UA project, see Section 7) of the security mechanisms they have designed (not implemented!) for their project. This milestone **does not involve any grading**, and will serve to assess the progress of students and to give them some feedback. Since the feedback will be given based on the written contents, they must be stored in CodeUA a couple of days before the milestone date. Please send an **email to the course teacher** upon the delivery of the overview.

7 Delivery Instructions

You should deliver all code produced and a report before the deadline. That is, 23.59 of the delivery date, December 31, 2018. Penalties will apply to late deliveries (1 point per day).

In order to deliver the project you should use a project in the CodeUA[3] platform. Please send an email to the course professor (André Zúquete) with the email of the group members to request a project creation. Do not create a project by your own!

Each CodeUA project should have a `git` or `svn` repository. The repository can be used for members of the same group to synchronize work. After the deadline, and unless otherwise requested by students, the content of the repository will be considered for grading.

[3] `https://code.ua.pt`