MIECT: Security 2019-20

# Secure, multi-player online Bingo game

Delivery of project code: after the exam on September 29, 2020
Deadlines: September 28, 2020 (inclusive)

André Zúquete

## Changelog

- v1.0 - Initial Version.

## 1 Introduction

The objective of this project is to develop a system enabling users to create and participate in online games (in this case, a sort of Bingo game).

The Bingo game that should be implemented should work (conceptually) as follows:

- Each player picks one or more cards. Each card is formed by $N$ distinct, random numbers, belonging to the set $[1, M]$ (for simplicity, we will refer to it as a **number deck)**;
- The game manager randomly selects (distinct) numbers from the deck until the victory of a player;
- The game winner is the first player that matches all the numbers of one of its cards to the numbers picked so far by the manager.

To make it more operational, we will change the way the game is played, but without changed its outcome:

- Each player generates randomly their cards, which should contain $N$ distinct numbers from the number deck;
- Repeated cards may coexist in the same game run. In case of victory, the total prize is split in equal parts;
- Everybody, game manager and players, shuffle the same number deck without having the opportunity to force a particular order. The final

(and random) order of the numbers in the shuffled deck gives their selection order by the manager.

The system is composed by a Bingo manager and an unlimited number 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 the manager using strong identification methods. The manager will only start the game with properly authenticated players.
- **Confidential and random number deck shuffling**: the manager creates a number deck and initiates its shuffling by the players. The shuffling should involve all players, and all of them should contribute to the randomness of the shuffling and to the guarantee that no single entity is able to know, for sure or with a high success probability, the order of the numbers in the shuffled deck.
- **Honesty assurance**: players must play with the cards they have picked (or created, in our case), and cannot use any other. To enforce this, players must commit to their cards before the start of the game. The commitment cannot reveal the cards (to prevent other players from forcing the occurrence of equal cards) and must be properly signed.
- **Correct game outcome**: the game evolution should be orchestrated by the manager. At the end of the game all players should compute the winner and detect cheaters (e.g. players with illegal cards or a manager reporting a wrong winner). Protests can be created to apply for third-party validations.
- **Correct accounting**: at the end of each game some accounting must be done relatively to the outcome of the game (e.g. if each player payed $X$ Euros for each card, the winner should earn $P \times X$ Euros). Winners should get accounting receipts signed by the manager in order to claim its amount in the future.

## 2 Project Description

### 2.1 System Components

We can consider the existence of two main components: (i) several players, all using the same exact code, and (ii) one server, which serves as manager for all players to connect to. A manager can serve only one game at the time. A game is started by giving a direct order to the manager. Players should not have to get any input from their users, other than the number of cards to play with.

### 2.1.1 Manager (server)

This manager (server) will expose a connection endpoint through which the players (clients) can exchange structured requests/responses with it.

The manager is the system component that gathers the players, creates the number deck, initiates the deck shuffling (with the help of the players), collects and distributes information from and to the players, in order to allow them to have all the information they need to reach a correct decision regarding the winner, broadcasts its decision regarding the winner, makes the game accounting and provides accounting receipts to players.

### 2.1.2 Player (client)

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

- Prove their identity (to the manager and to other players);
- Commit to secret information disclosed to others (cards, cryptographic keys, etc.);
- Produce a complaint.

Throughout the interaction between a player and a manager, or even an indirect interaction with other players, 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 manager 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 a manager (initiate the participation in a game);
- Commit to a set of cards;
- Shuffle the number card deck;
- Check the cards used by others;

- Select a game winner and check it against the manager's decision;
- Produce a complaint;
- Store accounting receipts.

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[1] 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 colluding entities, involving the manager, can manipulate with complete precision the order of numbers in the shuffled number deck in order force a given winner.

Decisions about winners should be taken by the manager looking solely to the information that was shared with the players during the game. Players that do not agree with the manager (it can cheat as well) should create a complaint (a signed game report record) with all the information allowing a third party to check the manager's decision.

Cheating players cannot benefit from it other than creation a Denial of Service scenario. However, the manager could remember the identity of cheating players, in order to prevent them to play again for some time. However, you are not required to do nothing in particular relatively to players' cheating other than detecting it and not considering it as a winner in any case.

## 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 set of cards is a computation performed over that set of cards which can be revealed before revealing the information that allows anyone to compute the winner: the shuffled number

[1]`https://en.wikipedia.org/wiki/Protocol_Buffers`

deck and all the players' 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 (which contains the original data from their set of cards), thus enabling others consider them when computing the game winner.

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 $C$ (set of 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 game protocol run involving a number deck randomly shuffled by all the player is not trivial, but there are some solutions. One is the following, that runs in four stages:

1. Commitment stage: this stage can occur immediately after the identification of the player by the manager. The player creates a set of cards (the exact number can be provided by the person running the player software). Note: players are not required to use random cards; they can use fixed cards for fetishism reasons.

   From the data of the set of cards, the player produces a single bit commitment and uploads it to the manager.

2. Preparation stage: when the game starts, the manager broadcasts all the bit commitments of all the players to all the players. This will allow players to validate others' cards when computing the winner.

3. Shuffling stage: the number deck with $M$ distinct numbers is created as a set of $M$ integer elements. Each player receive the deck at the time from the manager, shuffles the numbers randomly and applies an encryption per number (the same key, per participant, must be used for all numbers).

   The final, randomly shuffled number deck is broadcast to all the players, signed by the manager. Each number should have been ciphered as many times as the number of players. No player knows were each number is, so they cannot know the order of the numbers in the shuffled number deck.

4. Revelation stage: a player, after getting the signed and shuffled number deck, reveals the secrets it holds to the manager. Once having all the secrets from all the players, the managers broadcast them to all the players. With them, both the manager and the players can independently check the validity of the data they have (card sets from other

players, bit commitments, etc.), decrypt the numbers in the shuffled number deck, validate its integrity ($M$ integer values, values not lower than 1 or higher than $M$) and compute the winner. If everything was properly done, all players and the manager should reach the exact same decision.

## 4 Suggestions

The communication between players and the manager, or with other players, is easier to implement with TCP streams, one per player with the manager.

Most of the time the action of the players consists on receiving orders (or data) from the manager and to respond accordingly. For instance, produce a set of cards and provide its bit commitment, shuffle the number deck, reveal secrets, compute winner, receive winning receipt. Thus, players are clients of a server (manager) that most of the time are polled by the manager. This should not be an implementation a problem.

In order to enable the system to be tested with many 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.

Take care about the difference between the protection of messages versus the protection of special objects that can be a part of a message. Namely, signatures should be applied to self-contained objects that make sense by itself, and usually this does not apply to protocol messages. For instance, a receipt is a signed object that can be part of a message, but the signature assures the correctness of the receipt, and does not need to ensure the correctness of the message used to send the receipt.

## 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 manager with their Citizen Card;
- (2 points) Number deck secure shuffling;
- (1 points) Revelation of secrets;
- (2 points) Distributed game data validation;
- (1 points) Distributed winner computation;

- (1 points) Possibility of cheating;
- (2 points) Protest against cheating;
- (2 points) Accounting receipts;
- (1 points) Game accounting.

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.
- Consider the possibility of using secure sessions between players and a manager in order to reduce the cost of producing and validating digital signatures on messages.
- Involve the Citizen Card in all the required steps, including the validation of certification chains.
- Add bit commitments and security to the number deck shuffling protocol.
- Add the protest against cheating (consider this only when performed by a manager).
- Add cheating support.
- Add the production of receipts.
- Add accounting.

## 7 Delivery Instructions

You should deliver all code produced and a report before the deadline. That is, 23.59 of the delivery date, September 28, 2020. Penalties will apply to late deliveries (1 point per day, computed by the minute).

In order to deliver the project you should use a project in the CodeUA[2] platform. Please send an email to the course professor (André Zúquete) to request a project creation. Do not create a project by your own!

Each CodeUA project should have a `git` or `svn` repository. After the deadline, and unless otherwise requested by students, the content of the repository will be considered for grading.

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