Problem 3: Distributed Coin Flipping Game Using Stellar

Due: 11:59pm, Sunday, April 23, 2017

I. OVERVIEW

The objective of this course project is to use **Stellar** to build a Distributed Online Coin Flipping Game, which allows two or more participants to play the game fairly and transfer funds to each other safely. Stellar is an open platform for building financial products that connect people everywhere. The goal of Stellar is to achieve fast, reliable and near-zero-cost transfers. By using the services provided by the stellar platform, we can build a distributed payment system to support online Coin Flipping game.

II. PROJECT REQUIREMENT

This course project consists of three phases: Phase I project aims to develop a distributed payment system using Stellar, which allows users to send and receive funds between or among each other; Phase II project aims to develop a simple two-party online Coin Flipping game; Phase III aims to extend the system of Phase I and II into supporting multi-party online Coin Flipping game.

A. Basic Requirements

This is a project that needs to be done individually. The basic requirement is to finish both Phase I and Phase II of the project.

B. Extra Credit Opportunities

In addition to the above basic requirements, you can receive additional credit (5 points, on top of the 10 points for the basic requirements in Phase I and II) for Phase III of the project by implementing the required extensions:

The details about Phase I, II, and III requirements of this project are described as follows:

III. PHASE I

The first phase of the project is intended to familiarize you with the Stellar platform and learn the process of building user accounts. Most distributed applications interact with the Stellar network through **Horizon**, a RESTful HTTP API server. The detailed information of **Horizon** can be learned by following the links in Section VII. In this phase, you will create two accounts, with which users transfer lumens (funds) to each other. The basic requirements are:

1. Once the account is created, a new User Interface will pop up to allow user to query the account ID and its balance through that user interface.
2. If the user wishes to initiate a transfer to the targeted account, she can enter the address/accountID and transfer some amount of funds to the targeted account.
3. When user wants to check the activity history, the system can provide the transaction records within a day.
4. Errors, server exceptions, and invalid user input shall produce reasonably informative error descriptions to the user.
5. Learn the APIs that allow you to program with Stellar.

IV. PHASE II

Congratulations! If you have made it to this point, you should now have some working accounts and working knowledge to work with Stellar. Let us try something new to use Stellar! In this phase, you are expected to implement a two-party distributed Coin Flipping game. We start by a simplified "coin flipping" game: Alice and Bob want to bet ten dollars. They both agree to the bet ahead of time and the method for determining the winner. Bob will flip a coin in the air, and while it is rotating Alice calls out "Head" or "Tail". When the coin lands, they both immediately have a clear understanding of who won the bet, and they both have assurance that the outcome was random and that neither of them was able to influence the outcome.

One shortcoming is that the sequence of steps in this ceremony requires that both parties have to be present at the same place at the same time. Also, both parties still have to trust that whoever loses will pay up. In this

phase, we would like to be able to have an on-line coin flipping game that is not only just as "fair", but also the problem of making sure that the loser actually pays, with the support of implementation of **Stellar**.

The first challenge is replacing the "coin flip" mechanism with some online equivalent. Let's say we now have two parties, Alice and Bob, who all want to select a number with equal probability. Here's one attempt at such a protocol. Each of them picks a random number, e.g. Alice chooses 0, Bob chooses 1. They tell each other their numbers, and they combine the output as $val = 00, 01, 10$ or 11. Alice (first participant) wins if $val = 00$ or 11 (i.e., two numbers are the same). Otherwise, Bob (second participant) wins (if $val = 01$ or 10 (i.e., two numbers are different)).

If both of them chose their random numbers independently, this would indeed work. But remember that we are doing this over the Internet, and there is no way to ensure that they all send their numbers "simultaneously". Alice might wait until she hears Bob's numbers before broadcasting hers. If she does this, you can see how it is trivial for her to make the final output be whatever she wants. We cannot design the protocol to convince every party involved that none of the other parties can cheat.

To solve this problem, one possible solution is that we can use hash commitments by following the two-round protocol below:

Round 1: Each participant chooses a random number by calling appropriate random number generation function (most program languages like C and Java provide such functions). For example, Alice first picks x and Bob picks y, independently. Then each participant feeds her/his number to a hash function and gets the corresponding hash value, e.g., $H(x)$ and $H(y)$. After that, the hash value and her bet (e.g., 20 lumens) should be sent to an independent 3^{rd}-party banker. The banker will collect and keep the hash values and the bets deposited from each participant as well as the transaction time.

Round 2: The two parties reveal their values, x and y, to each other and the 3^{rd}-party banker. If both x and y revealed by the participants are consistent with the committed $H(x)$ and $H(y)$ in Round 1, respectively, the two parties and the banker can each decide who is the winner of the game by calculating $((x+y) \mod 2)$. If the result is 0, Alice is the winner. Otherwise, Bob is the winner. Doing so will guarantee no one can cheat, assuming we use a reliable hash function.

At the end of the game, 95% of the deposits from both parties will be sent to the winner from the banker, the remaining 5% will be retained by the banker as a service fee. This round of the game will end. Such game can be repeated many many rounds.

Requirements:
1. Please explain/prove whether this protocol actually works or not, in term of fairness, and what are the possible attacks (cheating) among the three parties. In your implementation, how have you solved them? You need to answer this question in your design document.
2. Implement the banker program that can store and update the game transactions. When the banker wants to query the activity history, the system should provide the game transaction records within one day. If some of the participants want to query the activity history, the banker needs to send the transaction record from previous round to each participant and allow them to verify the results.
3. You need to implement the hashing process in Round 1 yourself using any hash function like Double-SHA256, SHA256, MD5, SHA1, etc. and use appropriate random number generator function.
4. There are two user roles: participants and banker. A separate User Interface is required to each role/particpant. Only one active game at a time needs to be supported. At any given time, there may be at most one banker and at least two participants active with respect to a given round of game.
5. You can build a single application that supports both banker and participants interfaces, or two separate applications may be built, one for a banker and another for a participant. The banker's ID is public to each participant by default.

V. PHASE III

In this phase of the project, you will extend the two-party online coin fliping game into a multi-party game. In this phase, the number of participants may not be fixed. The banker starts a game at a particular time, and any number of participants can join the game and deliver their bets by a specified deadline (date and time). The banker needs to record the order in which each participant in the game and uses it as the ID of each participant in this round of the game. For example, Alice is the first person to join the game, then she has the ID 0. Bob and Carol are the second and third person to join this gambling, then their IDs are 1 and 2, respectively. Other participants are identified using the IDs generated in the same manner. The betting process is similar with that in Phase II:

Round 1: Suppose there are n participants in the current round of game, each of them independently chooses a random number and calculate its hash value. Then each participant needs to send the hash value and their bets to the banker. After the banker receive these info (i.e., commitment) will assign the ID for each participant,

depending on their order to join the game. For example, Alice first picks m_0, compute $H(m_0)$ and send the hash value and the deposit (her bet, say 20 lumens) to the banker. The banker receives the bet and hash value from Alice and assign the ID number 0 to Alice. Then Bob, Carol, David, and other participants of the game will get the ID number 1, 2, 3, respectively.

Round 2: Each participant reveals her/his random number m_i where $0 < i < n$, and send it to the banker. The banker collects all the m_is and check if all the m_is revealed by the participants are consistent with the committed $H(m_i)$s in Round 1, respectively, all the parties and the banker can each decide who is the winner of the game by calculating $(\sum_{i=0}^{n-1} m_i \mod n)$. If one of the participants' ID is equal to the result, she/he will be the winner. At the end of the game, 95% of all deposits (bets from the participants) will be sent to the winner from the banker, the remaining 5% will be retained by the banker as a service fee. This round of game will end. Also, the banker should notify all the participants about that the game is over and who the winner.

VI. WHAT TO HAND IN

A. *Design*

Before you start hacking away, plot down a design document. The result should be a system level design document, which you hand in along with the source code. Do not get carried away with it, but make sure it convinces the reader that you know how to attack the problem. The **Stellar.org** maintains JavaScript, Java and Go-based SDKs for communicating with Horizon. There are also community-maintained SDKs for Ruby, Python and C#. You can choose your own language to complete this project. The source code and reports should be submitted through Blackboard.

Please schedule a time with the TA to show the project demo between April 24, 2017 and April 28, 2017.

B. *Measurements*

In order to compare the efficiency of your implementation, you will perform some measurements. I will leave it to you to decide what measurements should be done.

VII. Useful links

1. A short tutorial on **Stellar** system: *https* : *//www.stellar.org/developers/guides/get − started/index.html*
2. **Stellar** javadoc is available at: *https* : *//stellar.github.io/java − stellar − sdk/*
3. **Stellar** laboratory: *https* : *//www.stellar.org/laboratory/*