DSE Project Writup - RASP: Rocky Arena of Scissors and Papers

Leonardo Aoun Lucas Gauchoux Rémi Coudert

February 7, 2020

1 Introduction

For this project, we chose to implement the infamous game of Rock Paper Scissors Battle with multiple players. It is a simple enough game to not have to deal with an overly complicated game logic while still having to implement interesting decentralized functionalities.

The basis for this game for n players is as follows : each player is able to send a rumour containing a proposition of bet and a hidden move, and wait for an answer from another player for this bet and their own hidden shape. Players can either create open challenges or challenge specific adversaries.

This will create a game of n players with asynchronous duels involving (fake) money and (safe) betting. Thus we implemented a consensus to keep track of duel results and players' money as well as some cryptography to implement the commit protocol to ensure fairness between players.

The interesting parts for this project are the commit process with which we ensure that no one can cheat and steal money and the consensus protocol (using a blockchain) to keep track of balances and past matches.

2 Related Works / Tools we used

2.1 Decentralized Bets

There is already a few games running on some Blockchains such as the Ethereum. Notably cryptokitties or lesser known games like Virtue Poker as well as implementations of the lottery. In this project we are mainly interested in the betting functionality. This functionality is used in many games such as Black-Jack. More precisely we implemented a "fair" betting system. Building what is so called a fair system is the main motivation of implementing a betting functionality on a decentralized system rather than on a client-server traditional system.

A fair game is a game in which players cannot cheat and is fair in its rules and rewards. With our peerster implementation we allow user-to-user bet and ensure that all bets taken are honored by using the block chain to register all transactions. A Proof-of-Concept(PoC) has been realized by betdemocracy.org. As explained in this PoC, by removing the bookmaker, you don't play against the casino or the bank anymore but directly against other user. This enables you to set your own odds and to propose this bet to others users in the network. This makes makes it very convenient for our implementation because in naturally includes the duel idea of a Rock, Paper,Scissor game. The user that will challenge another player can set their own price and adversaries are free to take the bet/risk or not. As soon as a challenge is accepted the block chain is used to register the bets and no player withdrawal is possible.

2.2 Cryptographic Tools

Since bets and plays are user-to-user we don't have any trusted referee in the middle that can states which peers wins. We therefore need cryptographic tools such as private and public key to ensure we are indeed playing with the peer that started the challenge, known as digital signature and hash function to ensures integrity of the money bet for example. Moreover when playing with another peer we need a way to ensure to the challenger that we have choose a move(rock, paper or scissor), that cannot be changed, without revealing what move we choose. Otherwise winning is trivial for the challenger. For this particular aspect of the project see sections 2.3 and 6.

2.2.1 Hash Functions

As stated before we will use hash function to ensure integrity of the messages sent. Hash functions are also used by our game protocol to publish a transaction without revelling the move we choose. In order to ensure integrity the known method is to send the message plain text along with the hash of this plain text. Upon

receiving such a message the receiver can hash the plain text and verify that it correspond to the hash send along with the message. We will also use hash function to hide some data that is transmitted. This is part of the implementation of the commitment scheme explained below in the Fairness Ensurance with implementation details in 3.2. The way hash will be used in this part is by hashing the move choose by the attacker along with a nonce. Without the nonce the challenger would just have to hash the 3 possible move to find out the move played by the other peers. This nonce allows the action to remain secret until the nonce is revealed. This random number will be revealed only when the challenger chooses his move.

2.2.2 Private/Public key

We will use private and public key to ensure that we are communicating/playing with the person we agreed to. This process is known as digital signature. Digital signature is a mathematical scheme for verifying the authenticity of digital messages or documents. Each node will have at their disposal pair of public and private key. The public key of a peerster will be by definition public and available on the block chain when it joins the network. Each time node A wants to ensure node B that the message originates from him, it will encrypt the message to be send with it private key. Upon receiving the message, node B will be able to verify that it originates from A by decrypting it with A's public key.

2.3 Commitment Schemes

To ensure fairness we will use a commitment scheme. A commitment scheme is a pair of functions $Commit$ and $Open$, that have *hiding* and *binding* properties such that, with a random R.

$$Commit(X, R) = (c, k)$$
$$Open(c, k) = X$$

The hiding property means that one cannot recover X from c and the binding property ensure that it's not possible to generate k', c' such that $Open(c, k) = Open(c', k')$. It should not be possible for an adversary to recover X from c only.

Put simply, a commitment scheme permits to *commit* a value that cannot be opened by a peer, and cannot be changed, thus we will be able to exchange our moves without being able to see the opponent's move before sending yours, and without the possibility to change your mind.

3 Ensuring Fairness

The main prerequisite for a game of Rock Paper Scissors is that the players should play synchronously and simultaneously, to ensure fairness i.e. to be sure that no player can cheat during a round by choosing a move according to the choice of the adversary.

In the case of RASP, we cannot make players show their shape simultaneously. The obvious choice is to use some cryptographic primitive such as public key cryptography and/or a commitment scheme.

To do so, and following section 2.3, we chose to use the $SHA256$ hash function. Our commit protocol is (in theory) as follows :

$$Commit(Move, Nonce) = (Sig, Nonce) = (SHA256(Move, Nonce), Nonce)$$
$$Open(Sig, Nonce) = \text{Find the move that corresponds to the pair } (Sig, Nonce)$$

In reality, with our implementation, since messages are encrypted we can send the move to the adversary with the nonce for them to simply check with the signature (see end of section 6). Moreover, in the actual implementation, we added UIDs (unique IDs) to matches to prevent replay attacks.

This makes us able to send move that the sender cannot change (biding), and the receiver cannot open before they are supposed to (hiding).

4 Communication

In order to better understand the next section, we will introduce the two types of messages:

- Rasp Messages which are sent between Alice and Bob privately so gladiators in the Rocky Arena of Scissors and Papers don't accidentally hustle together. They include a game action or a challenge request or a challenge response. They will piggyback on the underlying Peerster Private Messages (or Rumors if the challenge request is open).

- Transactions which will be processed by the blockchain that can only include a game action, no need to spam everyone with Bob's small bets. They will use the same logic as Homework 3 by replacing the File struct by a GameAction. These GameActions will be aggregated into a ledger that contains for each head, for each player, their balance, and for each game, its state on the blockchain.
  Map: (Head) -> (Map: Players -> Balance, Map: Identifier -> State)

5 Processing of the Blockchain into a Ledger

The introduced commitment scheme would be useless if Alice, the challenge proposer and winner, cannot let everyone know she beat Bob. And she cannot just send a rasp message with the results to everyone because Bob is a bad loser, and Remi, the Referee would not accept to see Bob cry so he'd change the results when Alice is not paying attention.
This is why we need a common immutable ledger: the blockchain we implemented in HW3.

The easiest naive solution would be to just broadcast the rasp messages contents using Transactions since the signatures ensure authentication. However, as we said, Bob is a sore loser, the next time they enter the Rocky Arena of Scissors and Papers, he proposes a challenge to Alice but when she sends him her Rasp Defence he realizes her move was stronger and does not send a reveal so no one knows he had lost. We thus decided to consider the defender the winner until the reveal from the attacker. So when the attacker sends an attack, they hold some money for the bet, and if a defender replies, they would get it until the winning reveals takes back twice that money from the defender and grants it to the attacker.

The next round, Alice sends a Rasp Request trying to make amends, but Bob is so angry that he accepts the challenge, but does not send a Rasp Defence when Alice sends him the attack. He thinks he's being cheeky by making Alice hold a lot of money for nothing. And this is why Remi the Referee gave the ability for the attacker to cancel a game. This way, a justified and signed Cancel Transaction can help Alice avoid such pitiful bad players, or dropped packets.

Still trying to cheat his way to the Viridian Gym, Bob waits for an attack by Alice, defends with a Rock and waits for her Reveal Transaction. As soon as he sees that her hidden move was a Paper he publishes a Defence Transaction saying he had Scissors. Now it's just a game of popularity, but as everyone knows Bob is a sore loser and aggressive, they have no choice but to trust his word and Transaction against the Rasp Defence he had sent to Alice, even if she shows them the Signature. After all, it is only in the Blockchain that we trust. So now, when Alice wants to send a reveal, she should wait for any defence, the Rasp Defence or the Defence Transaction and include its signature in the Reveal Transaction, proving that Bob had sent her that move.

Hence, we now have the following Storyboard when any peer, like Remi the Referee, is processing the blocks into a ledger:

1. When seeing Alice's attack, he deducts <bet> Etherasps from Alice's balance. But first, he checks that Alice's signature is good and that both players have enough Etherasps in the ledger.

2. At Bob's TxDefence, he checks Bob's signature and that there exists indeed a match with the corresponding identifier coming from Alice. If the attacker was not Alice Remi discards the Transaction, else keeps it in the pending pool until more information arrives.

3. When he sees a TxReveal, again, he checks signature and checks that both the Attack and Defence are either in the ledger or also in the pending transaction pool. If Alice had lost, nothing happens but if she had won, he deducts <2 * bet> Etherasps from Bob and grants them to Alice. At this point, Bob could be in deep trouble and go into a negative balance. At this point he is considered broke and no honest player should accept any action from Bob. Some reveals or pending games could still be pending and he'll get out of this situation.

4. TxCancel: He checks for the existence of a corresponding attack or keeps it in the pending transactions, discarding it if the attack wasn't by Alice. He checks signatures. This transaction would grant Alice back her <bet> Etherasps and will refuse Defences or Reveals once the block is mined. However, if there is a TxDefence or some TxDefence arrives before the TxCancel is selected to be mined, the defence takes precedence. Preventing trollers from sending attacks and cancels at the same time.

5. A special TxSpawn means a player joined the network, this gives the included Public Key and Gladiator name pair a starting amount of Etheraps.

When a new fork becomes the longer chain, we don't want Alice to lose all the games she had won, so when switching to a new head, all the missing transactions are published again and added to the pending transactions if they don't conflict with the current ledger.

6 Cryptographic primitives and distributing secrets

To use the functionalities above we need a public key encryption scheme, a hash function and a way to distribute public keys to the peers as a way to sign the transactions.

We first need a way to sign messages. For this we chose to use public key cryptography and the RSA protocol, to sign messages with players' private keys such that receiving peers can verify that it comes from the one it's supposed to come from using public keys (authenticity).

Let K be a public key, and k a private key and let

$$C_K(.), Dec_k$$

be the RSA encryption with the public key K and the RSA decryption with the private key k respectively. In this case, C_K creates signatures that let us ensure authentication and integrity. Authentication comes from the fact that any peer receiving $Y = C_K(X)$ can compute $X = Dec_k(C_K(X))$ and ensure the message X comes from the owner of the public key K. Integrity can be preserved by a good choice of what's encrypted inside messages to ensure no one can modify the important parts of a message without habing to modify the signature as well (which should be hard).

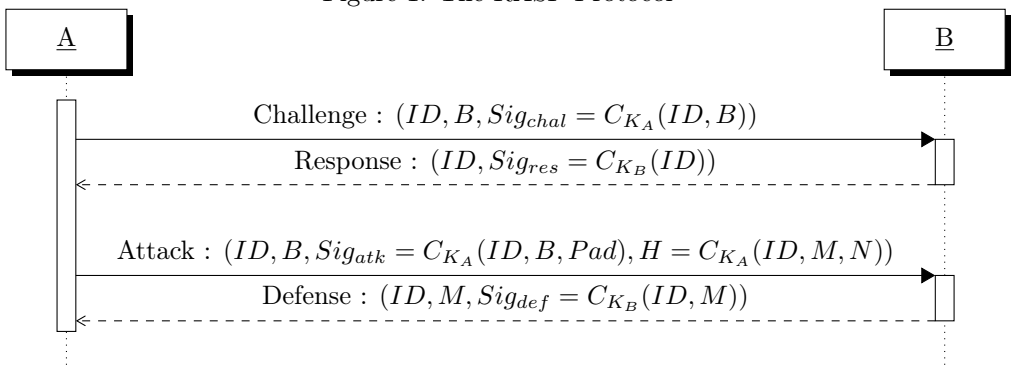
To implement this, all we need is to generate pairs of private and public keys for each peer in the network and each peer will need to broadcast their public keys. For this, we need to check the blockchain to find the public key that signed the arrival of a given peer, and store it locally.

We used golang's *crypto* package, that already implements hash functions, RSA and key generation.

It is important to note that *Go* encryption need messages to be of fixed length hence hashed, i.e. $C_K(X)$ is in reality $C_K(H(X))$. This is important to have in mind for section 7.3

7 Gaming protocol and implementation

Figure 1: The RASP Protocol



The main goal of the system is for people to be able to play Rock, Paper, Scissors with different players in the network such that they are able to bet money on each play.

The history of the win/loss transactions and the overall status of every player jackpot is ensured by the blockchain. The blockchain ensures that each players that bets money honors its commitment.

The whole protocol is depicted in Figure 1

Notations

In the following, we'll refer to A (Alice) the challenger and B (Bob) the Defender.

A given match in the game, is a round of Rock, Paper, Scissors played between A and B for some given amount of money (the bet).

We will refer as $C_K(.)$ for the creation of a signature, i.e. an RSA encryption with the public key K of the relevant player.

We'll refer as M for a move, and N for a nonce.

General overview of a single match

A match consists in 5 distinct parts :

1. A challenge sent by A
2. An answer to the challenge by B
3. The move played by A, signed
4. The move played by B, signed
5. The reveal, and the resolution of the winner (or draw)

The first two parts are used to create a match, and can be thought as something similar to a TCP handshake. The 3 later are the actual match, with the exchange of moves and the resolution of the winner if any.

A match is uniquely identified by its UID, and it can be cancelled. In the description of the different messages, we'll omit the destination and origin since it is not of interest of the actual protocol, but these fiels are used in the implementation.

Signatures

In most of messages, we use signatures to ensure authentication and integrity. Since signatures are RSA encryptions with public keys, it ensures authentication, and integrity comes from the way each message is designed (see below).

In the subsequent section, we'll only talk about integrity since authentication is always ensured the same way (see the section 6 on cryptography).

7.1 The Challenge

When A first decides to create a match, they decide on a bet amount B and a move M. There are two possibilities for a challenge :

- Open challenges: You send a challenge with your bet and it can be accepted by any peer that receive the message. Open challenges are propagated like rumours and the first node that reply will get the challenge.
- One-to-Ones: You can also challenge a known contact to a private battle.

Hence, a challenge has three components :

1. The match unique identifier ID
2. The Bet B for the match
3. The signature of the challenge identifier and the bet $Sig_{(chal,bet)}$

With

$$Sig_{chal} = C_K(ID, B)$$

The signature ensures integrity since an attacker cannot change the message without having to change the signature.

7.2 The Challenge Response

If B decides to answer to A's challenge (whether open or not), they send a response consisting of two components :

1. The match unique identifier ID
2. The response signature Sig_{res}

With

$$Sig_{res} = C_K(ID)$$

As for the Challenge, this signature ensure integrity.

7.3 The Attack

Once A and B agreed on playing a match against each other, and agreed on a bet amount B, A can send their actual move. This Attack message consists in four components :

1. The match unique identifier ID
2. The Bet B for the match
3. The bet signature Sig_{atk}
4. The hidden move H

With

$$Sig_{atk} = C_K(ID, B, Pad)$$

and

$$H = C_K(ID, M, N)$$

Sig_{atk} ensures integrity as before, and the pad (random, hidden and unique to a player) is used to prevent replay attacks (since the challenge signature would be identical otherwise).

H is the commited move, A won't be able to change it and B cannot verify it without having received A's move and nonce.

7.4 The Defense

B's defense has three components :

1. The match unique identifier ID
2. B's move (in clear)
3. the move signature Sig_{def}

With

$$Sig_{def} = C_K(ID, M)$$

Once again, the signature ensure integrity of B's move such that an attacker cannot change the said move without changing the signature.

7.5 The Cancel

As mentioned in the Blockchain explanation, Alice can send a Cancel if no one dares to challenge her unquestionable might. Or if some trickster like Bob tries answers her call but don't send any defence. She thus sends the struct:

1. The match unique identifier ID
2. The challenge signature Sig_{chal}, this time without the bet to avoid a replay attack.

8 Possible improvements, assumptions made and challenges encountered

There are several things to note :

- Players' balances can go (in a controlled way) in the negative, due to the possibility to accept multiple challenges in one block. This is intentional: We prevent someone from betting something larger than their balance, but we let them accept more challenges as long as they are in the same block. To allow some way for players to lose.
- We assumed peers to not be too malicious, e.g. they may want to try to find out opponents moves (hence the cryptography), but peers shouldn't try too much attacks such as denial of service or malicious messages. The only attacks they could do would be to change their own source code, but we made sure that following a different commitment scheme would result in either cancelled games or
- We also assumed that it is not our job to fix the issues with the Blockchain from HW3 (missing blocks when joining the network). This Rocky Arena of Scissors and Papers would run on stable Peerster blockchains just like Cryptokitties runs on the Ethereum network.
- The Peerster network must have a reasonable shape, no weak or missing links that would result in having two very different forks.
- We made sure that having malicious nodes would have the same effect as a a node that drops all packets.
- UIDs are supposedly unique if the system isn't used for too long, they are drawn independently at random in the range $[0, 2^{64} - 1]$.
- The processing of the blockchain is surely not optimal, sometimes we retraverse the whole chain and don't cache intermediate ledgers. We wanted to focus on having a good game and not focus on the architecture, so you cannot afford to take this code and host it on a Docker Swarm on AWS to run for a full year.
- To avoid a few potential issues coming from the Rewind problem: Imagine Alice cancels a challenge on the current HEAD chain, her TxCancel is there. Then there is a rewind and in the newer fork, Bob had sent his TxDefence. We made it such that anytime she sees a TxDefence concerning her and there is no TxReveal covering it, she will publish again its correspondent TxReveal.