REPORT

# Introduction

In our project we address the problem of how to implement a stateful multi-player game while minimizing trust in third-parties. By “stateful” we mean that there is some meaningful state that persists between separate interactions, as opposed to each interaction being selfcontained, as in chess, tic-tac-toe, etc. A typical example of a stateful game is poker, since each hand affects the money balances held by each player, which persist between hands.

Most games that exist today prevent cheating either by centralization — requiring that all interactions be routed through a trusted third-party solely responsible for maintaining the game state; or else by obfuscating the software running on each player’s device to prevent them from illegally altering the state.

In our project we seek a way to use cryptographic technology to alleviate these constraints. Our project implements a minimal game of this type as a proof-of-concept.

## Overview

For this project we have implemented a minimal example of a game with these properties, which we call “Rock-Paper-Scissors-with-state” (RPSWS). The rules of the game are as follows:

1. Any player may initiate an encounter with another player, who may or may not chooseto accept. (The two players are known as the *challenger* and the *defender* respectively.) No player may be in more than one encounter at the same time.
2. An encounter consists of a number of rounds. In each round, each player commits totheir move (either Rock, Paper, or Scissors), and reveals the commitment to the other player. The winner of each round is found according to standard Rock-Paper-Scissors rules (Rock beats Scissors beats Paper beats Rock).
3. The player to win two out three rounds wins the encounter.

## Architecture

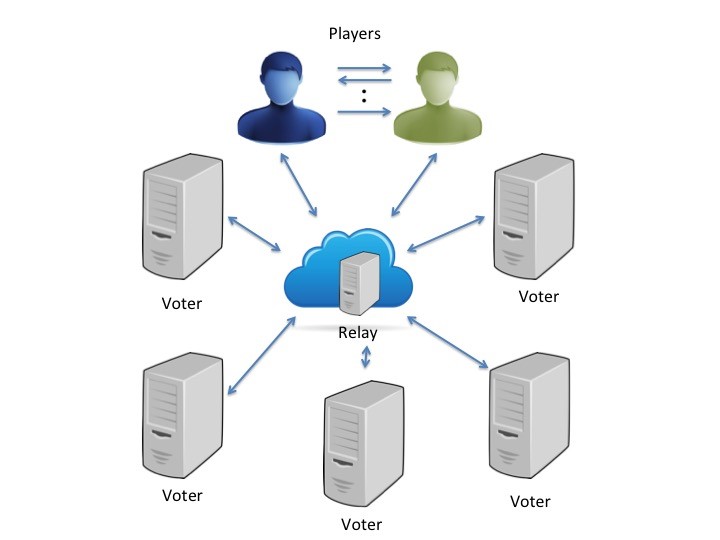


Figure 1: Overall Architecture

The game ecosystem as a whole may be conceptualized as a number of players who periodically interact with each other and with a “ledger” that represents the global state, available in common to all players.

The principle guiding our implementation is that each communication should involve as few parties as possible. Communication, when it does occur, goes over a peer-to-peer network for voters, and from point to point in the case of players. Players communicate with each other to play out encounters, which they then submit to the voters for verification and “disposition” (i.e. the process of updating the ledger to reflect the outcome of the encounter).

Accordingly, in our design we have taken pains to ensure that gameplay only requires the players to interact with the ledger at the beginning and end of an encounter. In particular, the players need not record their individual moves in the ledger as they make them; instead, the moves are assembled by the players themselves into a *encounter proof* transcript that they submit to the ledger at the end. Additionally, at the start, the players must query the ledger in order to record the beginning of their encounter; this prevents players from being in more than one encounter at the same time.

## Threat Model

Our threat model is concerned with the fairness of the gameplay and of the maintenance of the game state. We consider threats arising from both malicious players and malicious voters.

Players may exhibit various malicious behaviors during the game encounter, of which we consider three types. First, a player might attempt to change their move upon seeing the opponents choice; this is undesirable in RPSWS or any other game in which players are supposed to move “simultaneously” without knowledge of their opponent’s move. Second, a player may try to forge or replay in order to gain skill points without actually playing any games. Finally, a player may abort an encounter before a final outcome can be decided

(“ragequitting”).

The case where two players are both malicious and colluding is out of scope for our project; in any case, this scenario is unlikely to pose a threat in the context of an adversarial game.

We also consider the possibility of some amount of the voters being malicious and attempting to illegally alter the game state. They may attempt to alter account information or validate forged games in order to favor some players over others.

## Multiple games at once

We say that a player is trying to play multiple games at once, in the case when the player is trying to play another game while he is in the middle of playing a game, or has committed to playing one.

Let us assume that Alice is playing a game with Bob, or has committed to play a game with Bob. Alice’s current account state will reflect this fact – listing her as in encounter with Bob and that it started at a certain time. If Alice wants to initiate another game, the voters will check her current state and see that she is already committed to playing against Bob. A player is not allowed to play multiple games at once, so the voters will not include the initiate game request for Alice and the game will be discontinued.

# Implementation

We created three individual components as part of our software prototype: relay, voter, and client. For more information about the implementation and the code refer to Section 7.1

Relay instantiates a TCP server that voters connect to. By keeping a list of connected voters, it can relay a message received from any voter to all other voters to simulate an idealized lossless peer-to-peer network. In the current implementation, a client does not directly join the voter network, but instead posts and receives message from voters through a separate HTTP server interface on the relay.

Voter is a game proof verifier that records and maintains the global game state in a SQLite database. A voter upon receiving game proof from clients through relay’s HTTP interface, will validate the proof according to predefined rules and broadcast its validation result via relay to other voters. A voter also receives validation results from other voters. Once a consensus is reached, the SQLite database representing the ledger is updated accordingly. A voter must have a valid account already existing on the ledger to do so.

All components take advantage of asynchronous network programming featured in Python Tornado library. It abstracts low-level TCP networking API in a event-based paradigm and uses advanced Python coroutine feature to support concurrent access. This greatly eases the implementation of communications between components.

We use PyCrypto for the cryptographic primitives, including RSA encryption and digital signatures. For hash functions, we use Python’s built-in hashlib module.

Since no implementation for a commitment scheme was readily available, we created our simple commitment scheme. In our design, a user concatenates the value to be committed with a 20-byte randomly generated padding, and computes the SHA256 hash of the concatenated string as the commitment. To reveal the commitment, a user sends its committed value and previously generated padding to the verifier who computes the correct commitment and checks if it matches the previously received commitment.

We represent users in our system using their unique public-keys. In addition, similar to the Bitcoin network, we use a 48-character long account ID generated from the trailing 36 bytes of the hash of a user’s public key encoded in DER format. This significantly reduces the length of an account ID, while preserving the one-to-one mapping between users and account IDs. However, when users sign a message, they must include their public key along with the signature since the verifier may not necessarily have the signer’s public key for verification. The verifier must also check if the signer’s account ID corresponds to the received public key.

# Conclusion

We have designed and implemented Rock-Paper-Scissor-with-State to provide a verifiable, decentralized, game platform. Our voter network leverages existing consensus methods to allow for decentralized global state and our players provide verifiable game encounters. It is our hope that this general platform with its security guarantees and protocol flexibility can provide a base for more complex game protocols that include verifiable randomness, increased number of parties, and more efficient proof verification.

## Code Repository

Links to our implementation and information about usage can be found at the following locations:

* [Rock-Paper-Scissors-Cheat Github Repository](https://github.com/arihamlin/Rock-Paper-Scissors-Cheat)
* [Code usage](https://github.com/arihamlin/Rock-Paper-Scissors-Cheat/blob/master/README.md)