2020

# A Non-Interactive Zero Knowledge Proof for 3-Coloring the Petersen Graph

CS 295 – FINAL PROJECT WRITE-UP

MOLLY EATON

## Problem Statement

It is possible to adapt the previously implemented Interactive Zero Knowledge Proof for a 3-Coloring of the Petersen Graph to a Non-Interactive Zero Knowledge (NIZK) Proof. By adding shared randomness and allowing the Prover to package up a set of proof objects for the Verifier the element of interaction can be erased.

## Implementation

The original interactive version of this proof consisted of the Verifier alone choosing challenges to continuously send back and forth with the Prover. This can cause a number of logistical problems with communication, which is where the NIZK comes in. By eliminating the back-and-forth communication, the algorithm is able to increase convenience and efficiency. The main goal is to still keep the proof secure and reduce any chance of a cheating Prover getting away.

The Fiat-Shamir Heuristic allows the Prover to create its own challenges using a method of shared randomness. By hashing the commitment with a previously agreed hashing algorithm both parties can be satisfied that the hashed result will be random. Each hashed result is different with every round of the Prover essentially guaranteeing the Prover cannot deliberately choose a cheated edge.

This implementation uses the common SHA-256 hashing algorithm, the statement, to hash the commitment for each round along with the round's iteration number to add another layer of randomness for the hashed result. Each commitment holds the Prover's graph coloring with the colors themselves hashed so the Verifier cannot learn anything about the witness – the graph coloring. The quality of Zero Knowledge comes from the Verifier checking the algorithm without any knowledge of the witness. The hashed transformation of the commitment is then converted to an integer to perform modulo arithmetic to choose which edge becomes the challenge. The Prover opens the colors for this specific edge, alone, for the Verifier.

The above is run a certain number of times to create a series of proof objects for the Verifier. Each proof object contains the commitment, the challenge edge, the iteration number, and the colors of that edge. Once the proof objects are compiled the Verifier loops through and performs a number of checks to catch any cheating. First, the Verifier checks that the colors on the edge do not match (the main point of the 3-coloring). Second, the Verifier checks that the edge within the challenge matches which edge would have been chosen from the hashed commitment. Lastly, the Verifier checks that the colors released are that of the challenged edge.

After looping through all proof objects, the Verifier can determine with confidence whether cheating on the Prover's end was present, while the Prover can be assured that their witness is kept secret.

## Summarized Results

It can be said with gusto that this implementation satisfies the problem statement. The code implementation runs and works as expected. After running a number of rounds with the NIZK, all checks pass. The Verifier is able to trust the Prover with no interaction and the Prover is able to keep the proof Zero-Knowledge. These two main points are both parties' intention for a scheme such as this.