# CISC 878 Zero Knowledge Proofs Project

Nicholas Dionne (24kg1@queensu.ca)

## 1 INTRODUCTION

In this project, I will introduce Zero Knowledge Proofs (ZKPs) to a game of Battleship. Battleship is a relatively well-known strategy game where two players place ships on a grid and then take turns attacking the enemy grid in an attempt to hit their ships and sink them. This is typically done through some extensive guesswork until all of the enemy ships are sunk. The first Player to sink all the ships on an enemy grid wins the game. In a traditional game of Battleship, players have to trust that their enemy will correctly and accurately reveal when an attack hits or misses a ship on a grid.

### 1.1 Motivation

Fairness and privacy are important in a Battleship game. Battleship also relies on mutual trust, which can present some vulnerabilities in the game. For example, dishonest players can lie about the results of an attack to gain an advantage over their opponents. Due to this, players might have to show their current grid state and, therefore, ship positions to prove their honesty and the result of attacks; this, however, is detrimental to the game. In an online game format or even in a traditional game, Battleship can be supervised by a referee or third party, which enforces the fairness and privacy of the game. But this also leaves a large amount of private information about the game and its current state in the third party's hands to be potentially manipulated or shared, which is not ideal.

### 1.2 ZKPs for games

ZKPs help by offering a cryptographic solution to the above-mentioned problems. ZKPs allow players to prove that their statements, such as an attack hit or miss, are valid without revealing additional information, such as their current grid state and ship positions. This means that with each attack, ZKPs keep the relevant and vital information hidden while acting as a medium for proving and verification without the need for a referee, third party, or mutual trust in the game.

### 1.3 Project Objectives

The primary goal of this project is to understand ZKPs better and implement an interesting use of ZKPs (in this case, a ZKP Battleship game). Furthermore, during implementation, two differing ZKP algorithms will be used to improve our understanding further while also developing grounds for comparing the two algorithms chosen (in this case, Snarks vs. BulletProofs). In the implementation, I will attempt to establish a cryptographic method where players can commit their ship positions without revealing them at the start of the game. While also providing a proof system where players can verify their attack results without revealing ship locations. Moreover, the validity of ship placements and attack coordinates will be checked throughout the game to ensure fair play. A light comparison of Snarks and Bulletproofs based on my experience with implementation will follow all of this to check their suitability for games such as Battleship through the resulting proof sizes, verification speeds, and computational efficiency.

## 2 ZERO KNOWLEDGE PROOFS

Zero Knowledge Proofs (ZKPs) are a cryptographic protocol that allow a party, known as the prover, to convince another party, the verifier, that a statement is true without revealing any additional information through that process. This makes ZKPs particularly useful in cases that require privacy, such as blockchain transactions or, in our case, a game such as Battleship, where privacy is particularly important to help ensure a fair game. ZKPs can broadly be categorized into two types: interactive and non-interactive [9]. An interactive proof means the communication between the prover and verifier goes through multiple rounds which can make interactive proofs less efficient in some cases while non-interactive proofs allow for the prover to generate proof that can be verified with a single check instead thus removing the need for the extensive interaction found in traditional examples of ZKPs which are typically interactive [9]. Similarly there's also the concept of trusted setup where a list of private parameters are generated these have to be securely managed as if there compromised then the entire proof system is at risk and secondly there's non-trusted setup which is for ZKPs that don't require parameters which can make them more valuable for applications of ZKPs where a third party can not be trusted to hold the parameters [12]. Lastly, there are three key points to every ZKP: completeness, soundness, and zero-knowledge [3]. These can be defined as follows: completeness, if the statement is true, an honest verifier will be convinced; Soundness: A dishonest prover cannot convince the verifier if the statement is false; Zero-knowledge: No additional information is leaked beyond the validity of the statement [3].

### 2.1 Snarks

Snarks stands for Succinct Non-Interactive Arguments of Knowledge; Snarks is a rather widely utilized type of ZKP that allows a prover to convince a verifier of a computation's correctness [11]. Snarks have smaller, more minimal proof sizes and efficient verification times, as might be evident by the word succinct in Snarks, which makes them more suited for applications that need a faster means of verification [12]. In this project, Snarks will be utilized to implement a game of Battleship without the need for a referee (third-party verifier) or mutual trust. To be more specific, Snarks will be used to verify that an attack on the enemy's ships is either a hit or a miss in other words if an attack is verified that means it hit a valid ship position and if an attack is unverifiable by snarks it means an attack missed all valid ship positions. This will be done with the Halo2 library and by using Snarks, in Battleship players can commit their ship positions at the start of the game to be later verified during the game without leaking any information.

### 2.2 Bulletproofs

Bulletproofs are what we call a non-interactive zero-knowledge proof system [10]. Bulletproofs are designed to be relatively short, efficient, and they are made for range proofs without a trusted

setup [10]. This makes them also seemingly well-suited for confidential transactions or more privacy-seeking applications like most ZKPs [10]. In this project, Bulletproofs will be utilized in the implementation of Battleship to verify coordinate ranges, ensuring that ship placements and attacks remain within the bounds of the grid without revealing any additional information.

## 3 IMPLEMENTATION

This section is all about project implementation including both my experience as the programmer and any particular details found in the relevant references during project implementation.

### 3.1 Tools

The implementation of this project utilized many different and unique tools, the most important ones of which have been detailed below.

*3.1.1 Rust.* Rust is a rather peculiar programming language; it was chosen as the primary language first and foremost for its strong support of cryptographic libraries. In fact, in most cases, ZKP libraries seem to be implemented specifically for Rust or, in some cases, C++. In any case, Rust was chosen, and it seems Rust is also memory safe and has relatively good performance. Furthermore, again, as it needs to be said, Rust seems to have a multitude, if not the most, libraries and examples relating to ZKPs. If everything before wasn't enough, then there's also the rust analyzer, which gives great support when programming. Honestly, it has to be the best debugging tool I've personally used in any language, and just for that, I would recommend Rust.

*3.1.2 halo2.* The chosen library for my Snarks implementation was halo2, which is a Rust library developed by Zcash [8]. This library was made for constructing and verifying Snarks and from my understanding as well as first hand experience I can confidently say that halo2 was made in such a way that it could be as flexible as possible in circuit creation and allow for thorough customization that can increase proof generation and verification efficiency to the max [8].

*3.1.3 bulletproofs.* The library for the range Bulletproofs is quite fittingly named bulletproofs this library is however, supported further by merlin and curve25519-dalek for an implementation as defined and created by dalek cryptography [5]. The bulletproofs library provides Pedersen commitments for range proofs, and this is ultimately what will be used to verify that the coordinates are in the bounds Battleship game grid [5].

### 3.2 Snarks

The implementation of the project utilizes Snarks to prove that an attack in the Battleship game is valid without revealing any additional hidden information, such as the game board with all ship positions or individual ship positions. This proof system helps ensure that an attacker can verify their attack results and prevents any excessive cheating from their opponent, the prover. The circuit is represented by the BattleshipCircuit struct. This struct defines the secret ship positions, Attack coordinates, and hit results. Following that struct, things start to get a lot more interesting as we enter the circuit called BattleshipCircuit, which contains functions such as without witnesses, configure, and synthesize. The without witnesses function, from my understanding, specifies an empty circuit without witness values, which is required for the verifier's verification process [4]. The configure function basically sets up the circuit by laying out the columns in the Snarks constraint system, and in halo2, these columns are Advice columns: Store private (witness) values, Instance columns: Store public inputs, and Selectors: Control when constraints should apply [4]. The synthesize function essentially performs the actual Snarks circuit computation. It applies any constraints the function gets to the now defined values from configure and then applies those values to the following computation. In the case of ZKP Battleship, simple ship position hashes and attack hashes are compared to determine a hit or miss (true or false). Then the computed snarks result is compared with the claimed result of the attack using a constraint to help ensure proof results [4]. From here we leave the Snarks circuit behind, and we run into the generate proof function. This function quite literally just generates the proof with the required parameters, such as the pk, circuit, inputs, and writes them to a transcript with a bit of randomness from QsRng as well. Then we have verify proof strat this function just determines the proof verification strategy that Snarks will be using in this case it's a single verifier with the vk and the pre-generated params made in the trusted setup process. Write params is a function made to write the params.bin file as part of the trusted setup process required by Snarks this file is needed in many of the previously defined functions and is meant to be kept secure and secret, as misuse of the file can compromise the entire Snarks proof system. Lastly, we have the initialize params function, which utilizes the params file to set the necessary default parameters and generate the two ever-important keys pk (proving key) and vk (verifying key).

### 3.3 Bulletproofs

In this project, Bulletproofs are leveraged as a secure method for verifying that ship placement coordinates and attack coordinates remain within the bounds of the game grid. The smallest range size for a Bulletproof is $2^8 = 255$, which is a bit too large to use for a Battleship grid range check, so this is dealt with by utilizing an offset depending on the decided grid size. Otherwise, the Bulletproof only has a few more parameters, such as PedersenGens, which seems to be often used for committing values in ZKP systems, and BulletproofGens, which has a set capacity of 64 bits and specifies one party, or in other words, a single proof to be generated [5]. Following this, there's the blinding which generates a random blinding factor to hide the committed values from any possible observers [5]. Furthermore, similarly to Snarks, a transcript is made with all the relevant information, and then a range proof single prover is specified for proof generation, which should allow for the shifted coordinate (coord + offset) to be either verified within the range of $0 - 255$ or a failed verification indicating the coordinate is outside these set bounds [5]. Lastly, a new proof transcript is made, and then the proof is verified using verify single, and depending on the results, the proof will pass through using ok(()) or it will fail [5].

## 3.4 ZKP Battleship

ZKP Battleship adds zero-knowledge proofs to allow players to play the game without revealing the game board or ship positions to verify attack results or valid ship placements, and thus ruining the game. By utilizing Snarks, we can ensure that attacks are correctly verified against the committed ship positions, and through Bulletproofs, we can enforce that all attacks and ship placements happen within a valid range on the grid. These improvements to Battleship help remove one of the base flaws in the game, which is an over-reliance on mutual trust in the game's attack verification steps and setup phases. This, however, can somewhat be rectified by a third party making attack verifications and game setup checks, but this means every action taken by either player and even the setup of their grid's ship positions can be altered by the third party without the players' consent or even knowledge. Therefore, any third party has to also be highly trusted and secure, or the game remains compromised. Below are some images of the game in action in the VS Code terminal.

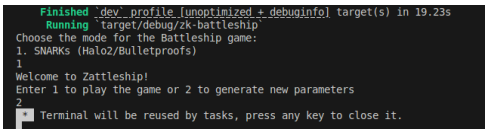


**Figure 1: params.bin Generation (Trusted Setup for Snarks)**

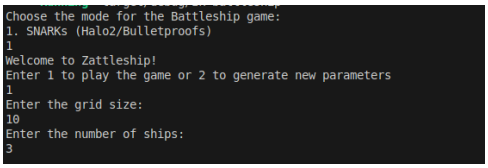


**Figure 2: Game Setup**

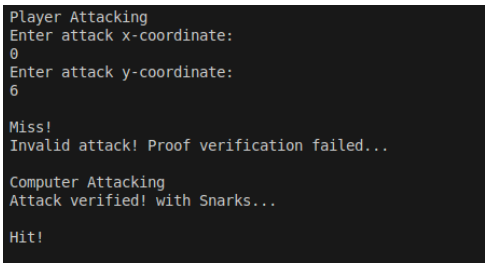


**Figure 3: Attacking/Game Flow**

The above Figures show the game-play loop in detail; however, keep in mind that the grids in the last two Figures don't hold the same ship positions as shown in the rest of the Figures, as they are different instances of the game. Figures 1 and 2 show the game start process, which is quite simple in Figure 1 a Snarks params.bin is generated to be able to use the Snarks proof system properly and in Figure 2 the general game start process is run through such as deciding the number of ships and the size of the grid which if you where to input 10 it would be a $10x10$ grid from $0-9$ for both the x and y axis. Then Figure 3 details the general repeated game flow of the player and computer attacking each other's grids in an attempt to hit an enemy ship. The computer launches attacks randomly while the player gets to input their attack coordinates. The only information shared at all is if the computer's or player's attack verifies and therefore hits a ship or fails verification and misses a ship. In the last two figures, we can finally see the player and computer's final grid state with "." representing dead water, "S" representing ship locations, and "X" representing sunken ship locations. These grids are only printed out in terminal to view at the end of the game when the computer or player has been announced as the winner.

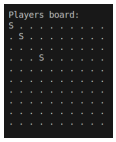


**Figure 4: Player Grid**

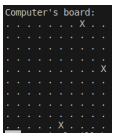


**Figure 5: Computer Grid**

## 3.5 Challenges

This project was overall quite challenging, both in the initial research of how ZKPs operate, as this can be difficult to wrap your head around, and in the implementation of ZKPs for the Battleship game. Some of the challenges I faced during implementation were understanding circuit design and halo2's general structure. These took quite a lot of time and patience to understand; however, the halo2 book and the Zordle game implementation were great resources that I consistently referenced to overcome this [4, 8]. There were also plenty of dependency issues, but I have to give full credit here to the Rust analyzer for VSCode, as the majority of my dependency-related issues were resolved just by implementing its recommendations. Lastly, I had originally attempted an implementation of Starks for the Battleship game to compare and contrast its effectiveness with that of Snarks however, the winterfell library for Starks was quite complex even more so than that of halo2's and quite frankly I could barely find anything substantial to reference in regards to implementing Starks with winterfell furthermore, I got stuck trying to implement the traces for the Starks proof system.

# 4 RESULTS

## 4.1 Game Results

The implementation of ZKP Battleship successfully produces zero-knowledge Battleship game play. Where the ships are both placed in valid positions and their positions stay hidden while attacks are fairly verified before the results are passed to players. Essentially, the results are that the game logic enforces attack verification without revealing valid ship positions.

## 4.2 Snarks vs. Bulletproofs

The table above offers a rather simplified breakdown of the significant differences between Snarks and Bulletproofs from proof size to scalability. The most interesting point mentioned above

| Feature | SNARKs | Bulletproofs |
|---|---|---|
| Proof Size | Small | Larger |
| Verification Time | Fast | Slower |
| Trusted Setup | Required | Not required |
| Scalability | High | Moderate |

**Figure 6: Table From Presentation (Comparing Snarks vs. Bulletproofs**

would likely be their differences regarding trusted setup. Bulletproofs don't require a trusted setup, unlike Snarks. This is a point of interest as trusted setup requires that the important required parameters be kept by a third party, which kind of defeats the purpose in a sense. As for the other differences shared in Figure 6 above, I would say that from the practical experience afforded to me during implementation that these are all relatively true, although I will point out that even though Snarks is faster in verification times, it's not by much.

## 5 CONCLUSION

This project helps demonstrate the feasibility of integrating zero-knowledge proofs such as Snarks and Bulletproofs into small games for the novelty of having a game such as Battleship that is now not reliant on mutual trust. By utilizing Snarks and Bulletproofs, the project ensures that fair gameplay while preserving the secrecy of the current game state is possible and does not overly affect gameplay other than minor proof generation and verification times, delaying actions. If I had to offer any suggestions for future work it would have to be firstly going back to complete that Starks implementation as the sense of defeat is a little frustrating otherwise, I would say exploring some additional cryptographic techniques among the project listings could be interesting or improving the efficiency of the proofs I currently have operating in the project to remove those slight delays [0].

## REFERENCES


[1] AdamISZ. 2023. From 0K to Bulletproofs. GitHub repository. https://github.com/AdamISZ/from0k2bp/blob/master/from0k2bp.pdf
[2] Rust Analyzer. 2023. Rust Analyzer. Project website. https://rust-analyzer.github.io/
[3] Baro77. 2022. ZKbasics Cheatsheet. https://github.com/baro77/ZKbasicsCS/blob/main/ZKbasicsCheatsheet20220621.pdf.
[4] Nalin Bhardwaj. 2023. Zordle. GitHub repository. https://github.com/nalinbhardwaj/zordle
[5] Dalek Cryptography. 2023. Bulletproofs. GitHub repository. https://github.com/dalek-cryptography/bulletproofs
[6] Rust Foundation. 2023. Cargo Documentation. Rust official documentation. https://doc.rust-lang.org/cargo/
[7] Rust Foundation. 2023. Install Rust. Rust official website. https://www.rust-lang.org/tools/install
[8] Zcash Foundation. 2023. Halo2 Concepts: Arithmetization. https://zcash.github.io/halo2/concepts/arithmetization.html
[9] Yannik Goldgräbe. 2023. Interactive Proofs and Zero Knowledge. *Medium* (2023). https://medium.com/magicofc/interactive-proofs-and-zero-knowledge-b32f6c8d66c3
[10] Stanford Cryptography Group. 2023. Bulletproofs: Short Proofs for Confidential Transactions and More. https://crypto.stanford.edu/bulletproofs/
[11] ZK Hack. 2023. ZK Hack Whiteboard - Module One. ZK Hack website. https://zkhack.dev/whiteboard/module-one/
[12] Phong Lee. 2024. Bulletproofs Without Trusted Setup. *DPle's Blog* (2024). https://dple.github.io/posts/2024/6/bulletproofs-without-trusted-setup/
[13] Sikoba Research. 2019. A Deep Dive into Bulletproofs. https://sikoba.com/docs/SKOR_DK_Bulletproofs_201905.pdf