

# *ZK-Junqi*: A Decentralized Chess Game with Zero-Knowledge Proof

Final Project Report  
Practical Cryptographic Systems  
EN. 601.445 / 601.645 (Fall 2021)

By  
Team - Xiecongyou Yang, Liyin Li, Yifan Wu, Meihan Lin

# Table of Contents

[Section 1. Introduction](#)

[Section 2. Related Works](#)

[Section 3. Relevant Preliminaries](#)

[Section 3.1 Junqi](#)

[Section 3.2 An Overview of Zero-knowledge Proof](#)

[3.2.1 Zero-knowledge Proof](#)

[3.2.2 Non-interactive Zero-knowledge Proof](#)

[3.2.3 Zero-knowledge Proof Integration with Circom and snarkjs](#)

[Section 3.3 Zero-knowledge Proof in Blockchain](#)

[3.3.1 An Introduction of Blockchain](#)

[3.3.2 Ethereum and Smart Contract](#)

[3.3.3 The framework of Zero-knowledge proof in blockchain](#)

[Section 3.4 Secure Multi-party Computation \(MPC\)](#)

[3.4.1 Overview and security definitions](#)

[3.4.2 Garbled circuit](#)

[Section 4. ZK-Junqi Development](#)

[Section 4.1 Application Architecture](#)

[Section 4.2 Application Implementation](#)

[4.2.1 Decentralized Technologies](#)

[4.2.2 Key Modules](#)

[Section 4.3 Game States](#)

[4.3.1 Game State 1: Initialization](#)

[4.3.2 Game State 2: Finish Setup](#)

[4.3.3 Game State 3: Move](#)

[4.3.4 Game State 4: Attack](#)

[4.3.5 Game State 5: End Game](#)

[Section 5. Application Findings & Analysis](#)

[Section 5.1 Advantages of Decentralization](#)

[Section 5.2 Challenges in Applied Decentralization](#)

[Section 5.3 Learnings & Future Works](#)

[Section 6. Conclusions](#)

[Citations](#)

[Appendix A.](#)

## Section 1. Introduction

The world has been operated by centralized systems for centuries. With the proliferation of cryptography technology, decentralized technologies provide new ways to organize information, of which the power is democratized to participants and thus individual privacy is better preserved. As an example, non-interactive zero-knowledge proofs (zk-SNARK) and secure multi-party computation (MPC) is a cryptographic protocol that achieves information secrecy and privacy respectively without the presence of a central agent. The former allows one to privately prove its possession of a secret information while the latter allows a party to jointly compute a function without revealing each other's inputs.

To practice these beautiful ideas in applied cryptography, we implement a decentralized peer-to-peer (P2P) online chess game, *ZK-Junqi*, with the integrations of MPC, zk-SNARK, and smart contract. As a redesign of the Chinese chess game *Junqi*, it does not require a third person besides the two players to act as a judge whose responsibility is to inspect opponents' hidden information and stimulate gameplay. Alternatively, we utilize MPC to enable players themselves to jointly compute the competing result with their hidden inputs, and delegate the arbitration to a smart contract program deployed on Ethereum server. The main deliverable of this project is a web application written in javascript, encapsulating the game logics with MPC's stimulation, zk-SNARK's validation, and smart contract's arbitration.

Quick Links of *ZK-Junqi*:

- Source Code: <https://github.com/xcyang/pcs-project>
- Application Demo: [https://drive.google.com/file/d/1AMkBETd4\\_4w8EmuHHTC\\_2XD0IrC29E1S/view?usp=sharing](https://drive.google.com/file/d/1AMkBETd4_4w8EmuHHTC_2XD0IrC29E1S/view?usp=sharing)

## Section 2. Related Works

Our implementation of *ZK-Junqi* is inspired by two other decentralized games: [Battleship](#) and [Dark Forest](#). Both games are integrated with zero-knowledge proof in realizing decentralization. We reference both works as our initial starting point. However, we encountered a particular challenge that cannot be solved by using zero-knowledge proof only, due to the nature of the game *Junqi*. As a result, we integrated the MPC protocol in resolving it. Details of the development process and learnings can be referred to Section 4 and Section 5.

## Section 3. Relevant Preliminaries

### Section 3.1 *Junqi*

*Junqi*, also named as *Luzhanqi* is a two-player Chinese board game. The ultimate goal of the game is to capture the opponent's flag (one of the pieces) while preventing own's is captured. Each player has a total of 25 pieces with an inherent rank attached accordingly. During gameplay, players take turns to conduct a *move* or an *attack*—reallocating a chosen piece to an empty spot or to an opponent's spot. Piece movement is restricted by the inherent rank as well as the featured barriers on *Junqi*'s game board.

The motivation of utilizing *Junqi* to practice the idea of decentralization. It has an unique, critical property that all opponent pieces' ranks except its locations are **hidden** to a player. The original *Junqi* needed a third person to act as a judge in facilitating the gameplay. When a player chooses to conduct an *attack* (reallocating its own piece to an opponent's spot), the judge needs to determine whether the attack succeeds by comparing the ranks of the two pieces. The one with a smaller rank will be captured and removed from the board, meanwhile a tie will contribute to a removal of both pieces. Most importantly, no rank information of opponent's is explicitly disclosed throughout the game. Detailed explanation of *Junqi*'s game board and pieces including the corresponding movement constraints are attached in *Appendix A*.

### Section 3.2 An Overview of Zero-knowledge Proof

In this section, an overview of zero-knowledge proof is demonstrated including its definition and construction. Particularly, the zk-SNARK, a non-interactive version of it is elaborated with an existing implementation written in `javascript`.

#### 3.2.1 Zero-knowledge Proof

A standard zero-knowledge proof is a cryptographic protocol by which one party, a prover, can prove to another party, a verifier, that a given statement is true without revealing any information beyond the fact that the statement is indeed true. As an example construction shown in *Figure 1*, the Prover attempts to convince the Verifier upon a proving statement through a three-way handshake:

1. **Witness Phase:** The Prover computes a proof that contains its proving statement and transmits it to the Verifier.
2. **Challenge Phase:** The Verifier asks the prover several questions.
3. **Response Phase:** The Prover answers these questions which can be used by the Verifier in considering whether to accept or reject the Prover's proof received at the witness phase.

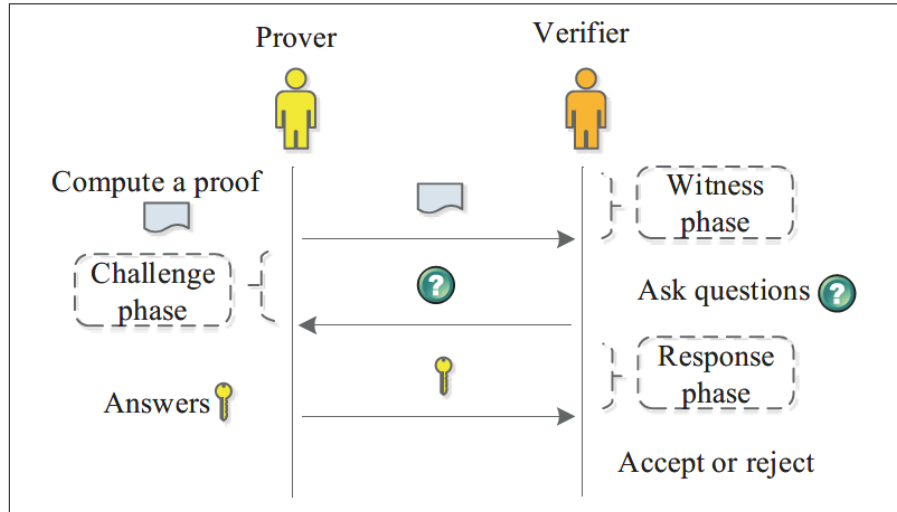


Figure 1. Shows an example construction of a zero-knowledge proof.

A zero-knowledge proof must satisfy three critical properties:

1. **Completeness:** The Verifier will eventually be convinced if the Prover is honest.
2. **Soundness:** The Prover can convince the Verifier if and only if the statement is true.
3. **Zero-knowledge(ness):** The Verifier learns no information beyond the fact that the statement is true.

### 3.2.2 Non-interactive Zero-knowledge Proof

As an optimization of the standard zero-knowledge proof, zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARK) is proposed by Bitansky et al in 2012. Its construction relies on three mathematical foundations: homomorphic hiding, quadratic arithmetic programs, and blind evaluation of polynomials (X, Battleship).

*Figure 2* demonstrates the flow of verifying zero-knowledge proofs in an zk-SNARK implementation. First, a proving key  $PK$  and a verification key  $VK$  are generated during setup, using a predefined parameter  $\lambda$  and an F-arithmetic circuit  $C$ . Both the input and output of Circuit  $C$  are elements in a field  $F$ . As for the outputs of setup,  $PK$  is used for generating verifiable proof, while  $VK$  is used for verifying the generated proof. Second, the Prover needs to provide the generated  $PK$  along with the Input  $x$  (could be private or public) and the Witness  $W$  in order to generate a proof  $\pi$ . Lastly, the verifier verifies  $\pi$  with the public portion of input  $x$  and determines whether to accept the  $\pi$ .

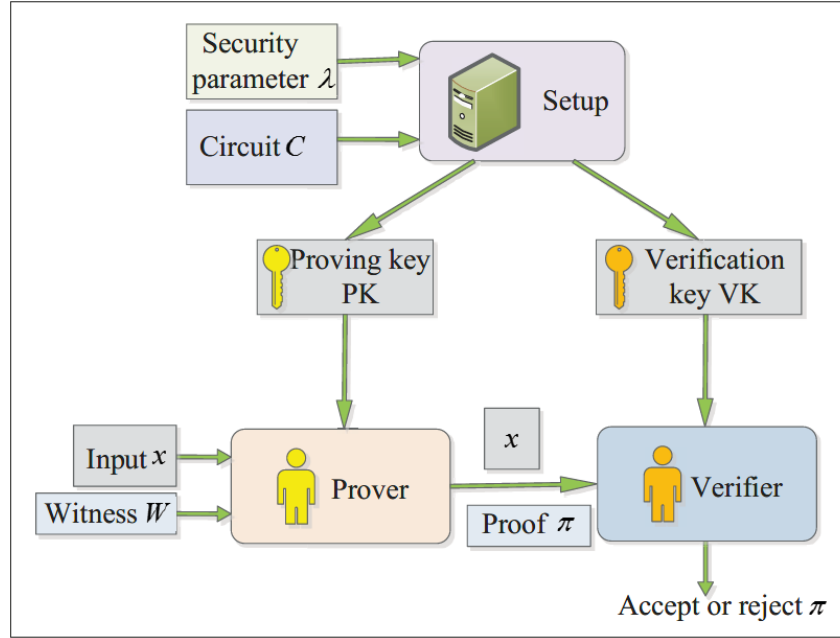


Figure 2. Shows the flow of verifying a zero-knowledge in an zk-SNARK implementation.

This implementation of zk-SNARKs contains several properties, making it more efficient in a real-world application such as blockchain in comparison to a standard zero-knowledge proof model. First, the verification process can be executed efficiently because the generated proof size is small, usually only with several bytes. Second, the process of proof sending from the prover to the verifier need not be synchronous. And the process of proof generation and verification can be done off-line respectively. Third, the proving algorithm is fast and bounded under a polynomial time. With these advantages, zk-SNARK is implemented in the Zerocash blockchain protocol.

### 3.2.3 Zero-knowledge Proof Integration with Circom and snarkjs

In order to develop customized zk-SNARK proofs for *ZK-Junqi*, two tools are needed: `Circom` and `snarkjs`. `Circom` is a programming language for arithmetic circuit construction. `Snarkjs` is an implementation of zk-SNARK with two alternatives, Pinocchio and Groth16 protocols. The former allows us to prototype desired constraints for our zk-SNARK proofs and the latter is used for translating circuit logic into executables that can be compiled by a decentralized machine for the process of proof verification. Besides, `snarkjs` has a command line interface (CLI) that allows developers to test proof generation and verification through terminal.

Figure 3 shows the construction of a `circom` circuit program that we used in *ZK-Junqi*. It is a deterministic program that contains input and output signals. As for its input, there are two types: private and public. Specifically, the private input is hidden from the verifier. The output of the `circom` circuits is fed into `snarkjs`. And the `snarkjs` program will complete the remaining steps in a zk-SNARK implementation.

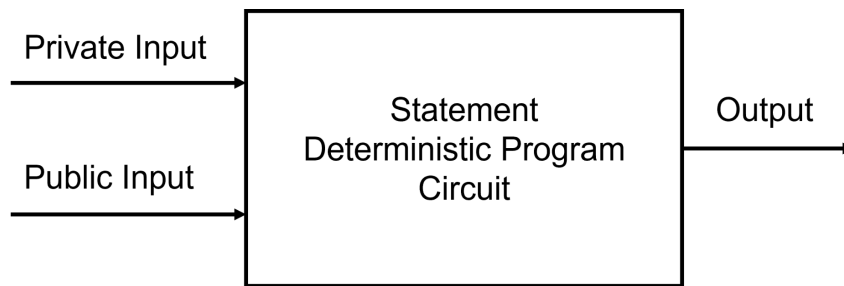


Figure 3. Shows the construction of a circuit program.

Given the output of the circuit program, `snarkjs` generate a proving key  $PK$  and a verification key  $VK$ . Apart from the two keys, it also generates a witness file of *wasn* type containing the private input of a proof. A proof object is created with the  $PK$  key and the witness input. This process also outputs a file with all the public inputs of the circuit program which will be needed in the proof verification process. Ultimately, the generated proof materials are submitted to a `solidity` program deployed in a decentralized server for verification. In achieving this, the generated proof has to be translated into a format that can be interpreted by a `solidity` compiler. Fortunately, `Snarkjs` supports such translation, of which a `solidity` program targeting the current circuit program can be generated through CLI.

## Section 3.3 Zero-knowledge Proof in Blockchain

In this section, we present a brief introduction of blockchain, one of its notable examples – Ethereum and smart contract, and a framework of zero-knowledge in the environment of blockchain.

### 3.3.1 An Introduction of Blockchain

A blockchain is a decentralized, distributed, and oftentimes public, digital ledger consisting of records called blocks that is used to record transactions across many computers so that any involved block cannot be altered retroactively. Its associated data written blockchain database is managed autonomously using a peer-to-peer network and a distributed timestamping server.

### 3.3.2 Ethereum and Smart Contract

Ethereum is an open-source public blockchain platform with smart contract functionality, providing decentralized Ethereum Virtual Machine (EVM) to handle peer-to-peer contracts through its dedicated cryptocurrency Ether. Ethernet provides various modules for users to build applications through a Turing-complete scripting language (Ethereum Virtual Machine Code, or EVM language).

The application described above is actually the smart contract, which is the core of Ethereum. A contract is an automated agent that lives in the Ethereum and has its own address. When a user

sends a transaction to the contract's address, the contract is activated, and then based on the additional information in the transaction, the contract runs its own code and finally returns a result, which may be another transaction sent from the contract's address. It is important to note that a transaction in Ethereum is more than just sending Ether, it can have quite a bit of additional information embedded in it. If a transaction is sent to a contract, then this information is very important, because the contract will use this information to complete its own business logic. In addition, smart contracts can reduce the need for trusted intermediators, arbitrations cost, enforcement cost, and fraud losses, which makes it an ideal model for developing decentralized applications.

### 3.3.3 The framework of Zero-knowledge proof in blockchain

The architecture of zero-knowledge proof in blockchain is shown in *Figure 4*. There are two parts in this architecture: the on-chain part and off-chain part.

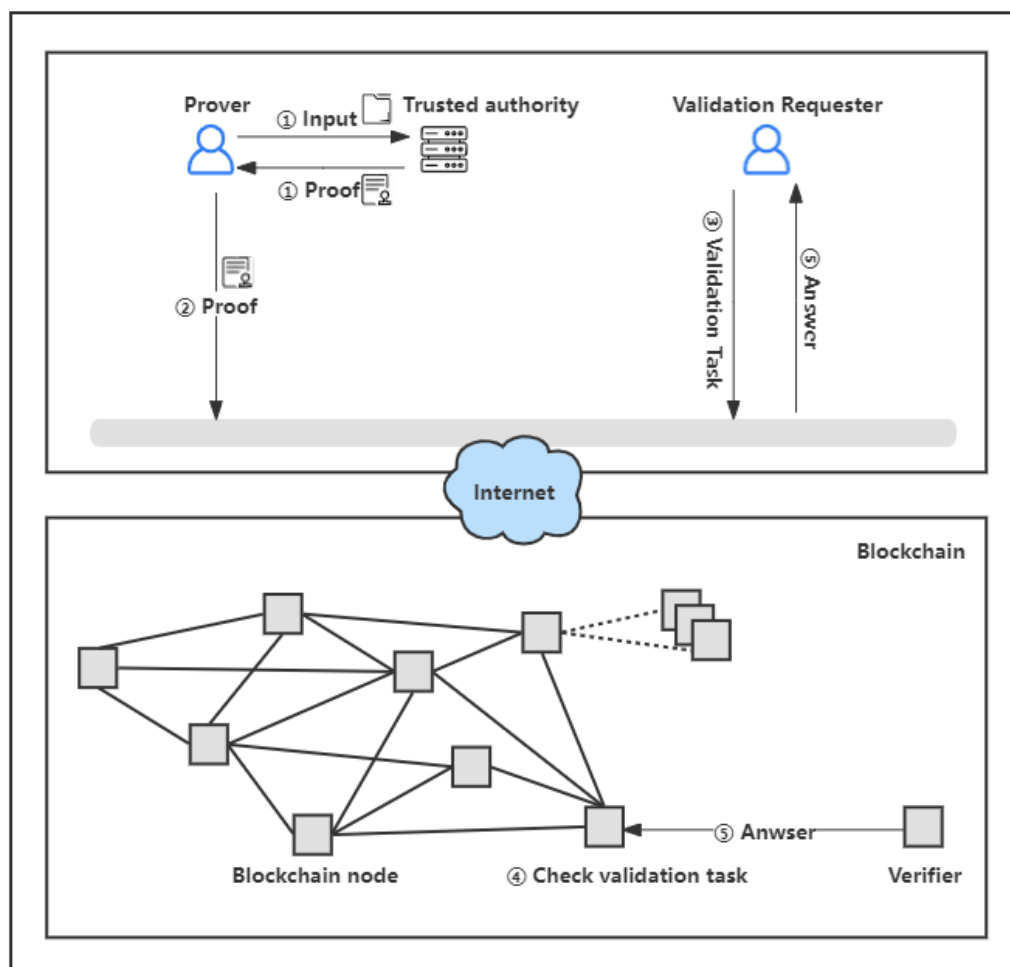


Figure 4. The architecture of zero-knowledge proof in blockchain.



- Off-chain, the prover claims a statement (e.g. he owns enough transaction amount). The validation requester is responsible for announcing a verification task, collecting the verification result from the verifier, and paying the verification fee to the verifier.
- On-chain, the authenticity verification of the prover's claim is implemented by the verifier, which is usually a blockchain miner. In addition, blockchain has an incentive mechanism, which calculates the verification fee for the verifier.

The streamlined implementation of this authenticity verification consists of the following steps:

1. Trusted authority generates the proof and transmits it to the prover.
2. The prover transmits the generated proof to the blockchain by the Internet. Then, the proof is stored on the blockchain, which can guarantee the integrity and non-tamperability of the proof.
3. The validation requester sends the verification task to the blockchain.
4. When the blockchain node receives the verification task, it checks the task and if valid, the verification request will be flooded to the verifier.
5. A verifier that is interested in this task will verify the proof and send the answer back.
6. The validation requester can confirm the prover's statement based on the accepted verification result.

Blockchain generates one or several new blocks, which can be used to record the process of authenticity verification without any falsification.

## Section 3.4 Secure Multi-party Computation (MPC)

### 3.4.1 Overview and security definitions

MPC enables multiple parties, each holding their own private data—to evaluate a computation without ever revealing any of the private data held by each party (or any otherwise related secret information). The two basic properties that a multi-party computation protocol must ensure are:

- **Privacy:** The private information held by the parties cannot be inferred from the execution of the protocol.
- **Accuracy:** If a number of parties within the group decide to share information or deviate from the instructions during the protocol execution, the MPC will not allow them to force the honest parties to output an incorrect result or leak an honest party's secret information.

The security definitions of an MPC are demonstrated below:

- **Real-Ideal Paradigm:** each participant is trustworthy, and one party sends its information to the other party, who does not look at the information, but only calculates the result according to the rules and sends it to the next party or to all participants.

- **Semi-Honest Security:** a participant will run the protocol honestly, but will derive additional information based on input from other parties or intermediate results of the computation.
- **Malicious Security:** a malicious model may not run the protocol honestly, and may even sabotage it.

In this project, the two-party secure computation (2PC) protocol is the MPC implementation that we used which involves two participants only. And the security assumption that we make for *ZK-Junqi* is of semi-honest.

### 3.4.2 Garbled circuit

Garbled circuit is a cryptographic protocol that enables 2PC in which two mistrusting parties can jointly evaluate a function over their private inputs without the presence of a trusted third party.

The protocol is constructed as follow:

1. Alice garbles (encrypts) the circuit. We call Alice the garbler.
2. Alice sends the garbled circuit to Bob along with her encrypted input.
3. In order to calculate the circuit, Bob needs to garble his own input as well. To this end, he needs Alice to help him, because only the garbler knows how to encrypt. Finally, Bob can encrypt his input through oblivious transfer. In terms of the definition from above, Bob is the receiver and Alice the sender at this oblivious transfer.
4. Bob evaluates (decrypts) the circuit and obtains the encrypted outputs. We call Bob the evaluator.
5. Alice and Bob communicate to learn the output.

The Garbled Circuit is secure against a semi-honest adversary. It is more challenging to make this protocol secure against a malicious adversary that deviates from the protocol.

## Section 4. *ZK-Junqi* Development

### Section 4.1 Application Architecture

*ZK-Junqi* is a P2P web application. Its architecture captured in *Figure 6* includes two main components: a web application containing frontend, ZK-SNARK, and MPC logics and a smart contract program deployed on Ethereum network. Specially, the frontend logics are written in `javascript`, and it allows players to interact and submit their game actions during gameplay. As for the other two important features in this web app, the zk-SNARK proofs generation is written in `circom` circuit language while the MPC computation with secret exchange is written in Golang.

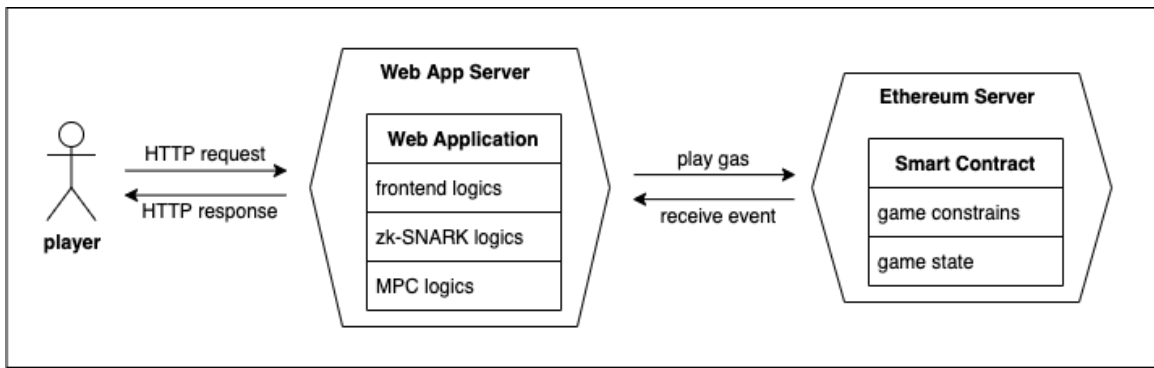


Figure 5. Shows the architecture design of the *ZK-Junqi* web application.

The implementation presented above is not ideal, not all computations are done in a decentralized manner. Both MPC and zk-SNARK's computations rely on the web application server. Due to incompatibility issues of `circom` and `snarkjs` in applied zk-SNARK, we cannot deploy the zero-knowledge proof at the client side. As for the MPC logic, we utilize a pre-implemented MPC program from Github instead of implementing our own version from scratch. Thus the deployment of the MPC program is only for a proof of concept. Ideally, the *ZK-Junqi* architecture should as *Figure 6* shown.

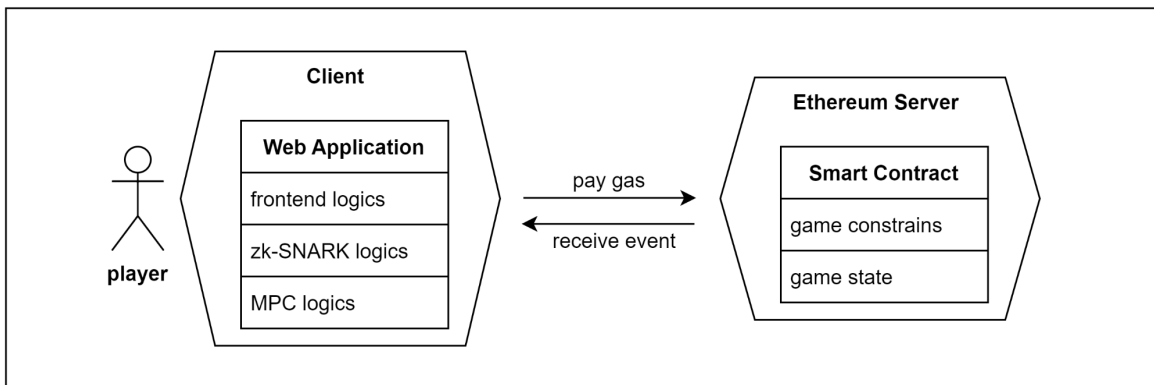


Figure 6. Shows the ideal architecture design of the *ZK-Junqi* web application.

## Section 4.2 Application Implementation

### 4.2.1 Decentralized Technologies

To achieve decentralized arbitration in *ZK-Junqi*, we rely on blockchain technology, Ethereum which operates on a decentralized computer network. It allows us to explicitly define desired game constraints in a smart contract program and deploy it over a public network. Each game movement can be regarded as a transaction from one player to another, Ethereum is the very role which validates the transaction for us. Even though the arbitration process will be justly done through executing the deployed smart contract, unfortunately it costs gas in each execution. Metamask is the tool that we use for managing the payment process triggered by every query to Ethereum server. For the ease of Ethereum development, we employ a development environment named Truffle Suit to deploy our smart contract and use private Ethereum blockchains for testing by Ganache.

### 4.2.2 Key Modules

The web applications *Junqi* includes two important modules: zk-SNARK and MPC. The zk-SNARK program is written in `circom` and `javascript`, and it is responsible for the integration of zk-SNARK proof. The MPC program is an implementation of the garbled circuit protocol written in `Golang`. It is retrieved from Github which can be found [here](#). Programming details can refer to our Github repository: [pcs-project](#).

## Section 4.3 Game States

*ZK-Junqi* supports multiple game instances simultaneously. For each game instance, there are five game states: *Initialization*, *Finish Setup*, *Move*, *Attack*, and *End Game*, as captured in Figure 7.

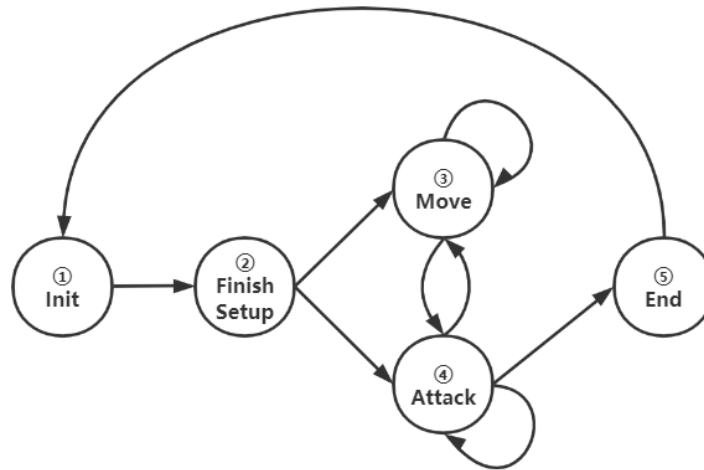


Figure 7. Shows the state diagram of *ZK-Junqi* which contains the *Initialization*, *Finish Setup*, *Move*, *Attack*, and *End Game* state.

#### 4.3.1 Game State 1: Initialization

When a player first enters the home page of *ZK-Junqi*, a window of MetaMask wallet will popup. This setup allows the program to store the player's ethereum address and hence enables Ether transactions between the client and the Ethereum server.

Once the setup is done, players can choose to start a new game or join an existing one. In a new game initiation, the frontend calls the method `declare()` defined in our smart contract program, in which a new game instance will be added to a global game list. As for the process of joining, the frontend makes another method call to `join()`. After the join is verified with sanity checks, smart contract will update the state of the joining game to be active along with the joiner's information. Whenever an execution of smart contract is triggered, the player will receive a payment request.

#### 4.3.2 Game State 2: Finish Setup

In the beginning of the game, players are required to set up their game board in compliance with the game rules. The board setup constraints can be referred to the **Game Board** section in *Appendix A*. After the boards are set up accordingly by players, zero-knowledge proof materials are generated at the client sides and sent to the smart contract deployed in Ethereum network as *Figure 8* captured for further verification process.

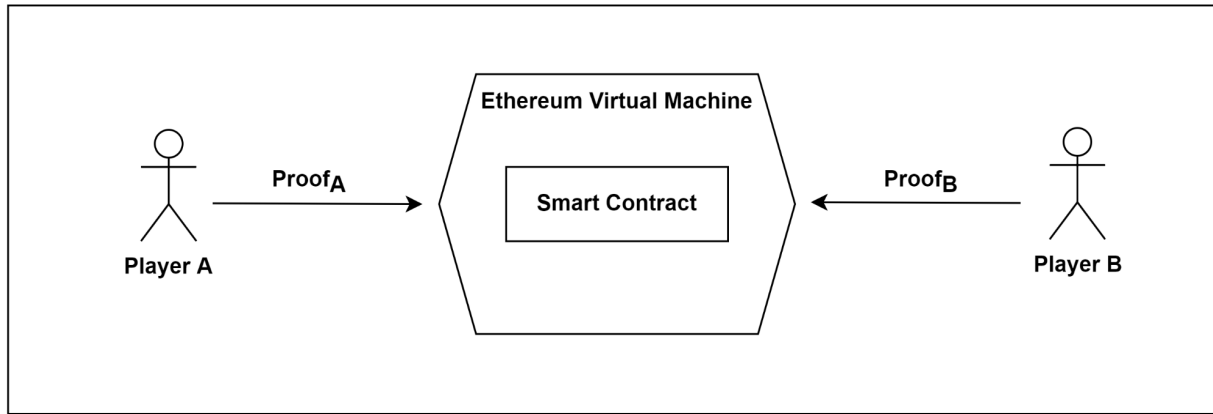


Figure 8. Shows players sending proofs to the smart contract program during *Finish Setup* state.

After the *Finish Setup* state, players can be transmitted to the *Move* or *Attack* state depending on the game actions players take.

#### 4.3.3 Game State 3: Move

When a player moves a target piece to an empty spot in the game board, it will be transmitted to the *Move* state. In the zero-knowledge proof that we construct, the player's move is validated by checking whether a reallocation from point A to B is in compliance by the rules determined by

the inherent rank of the target piece. *Figure 9* captures the interaction between the players and EVM during a *Move*.

- Player A sends its starting and ending positions,  $P_s$  and  $P_e$  to EVM along with its zk-SNARK proof.
- The information of  $P_s$  and  $P_e$  are broadcasted to player B.
- Player B sends its version of zk-SNARK proof of moving its  $P_s$  and  $P_e$  to EVM.

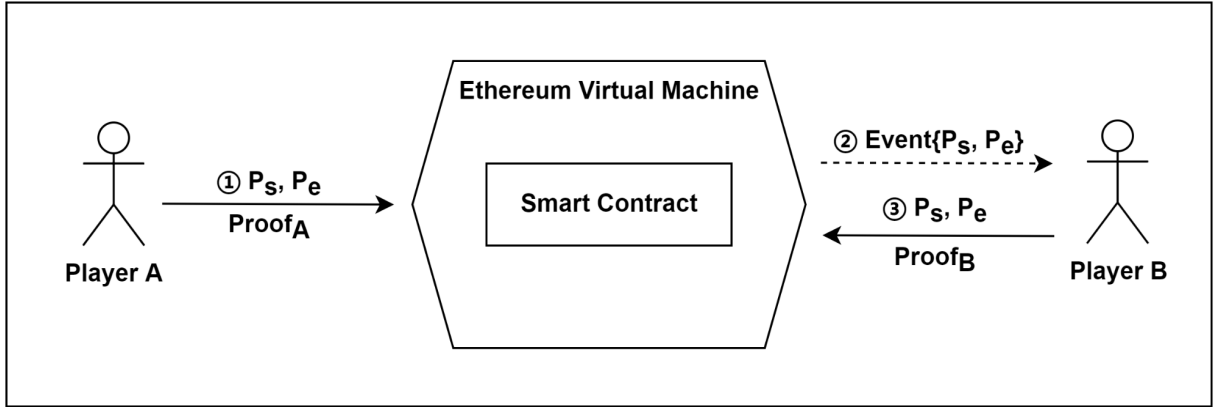


Figure 9. Shows players sending proofs to Ethereum during *Move* state.

#### 4.3.4 Game State 4: Attack

When a player moves a target piece to one of its opponent's spots, it will be transmitted to the *Attack* state. In the zero-knowledge proof that we construct for the *Attack* state, the rank comparison result obtained by both players also needed to be included apart from the location information. *Figure 10* captures the interaction between the players and EVM during the *Attack* state.

- Player A sends its starting and ending position,  $P_s$  and  $P_e$  to EVM.
- The information of  $P_s$  and  $P_e$  are broadcasted to player B.
- Player A and B compute the rank comparison with their respective secret input  $R_s$  and  $R_e$  using the MPC program.
- Player A and B send their version of zk-SNARK proof of moving their  $P_s$  and  $P_e$  to EVM.

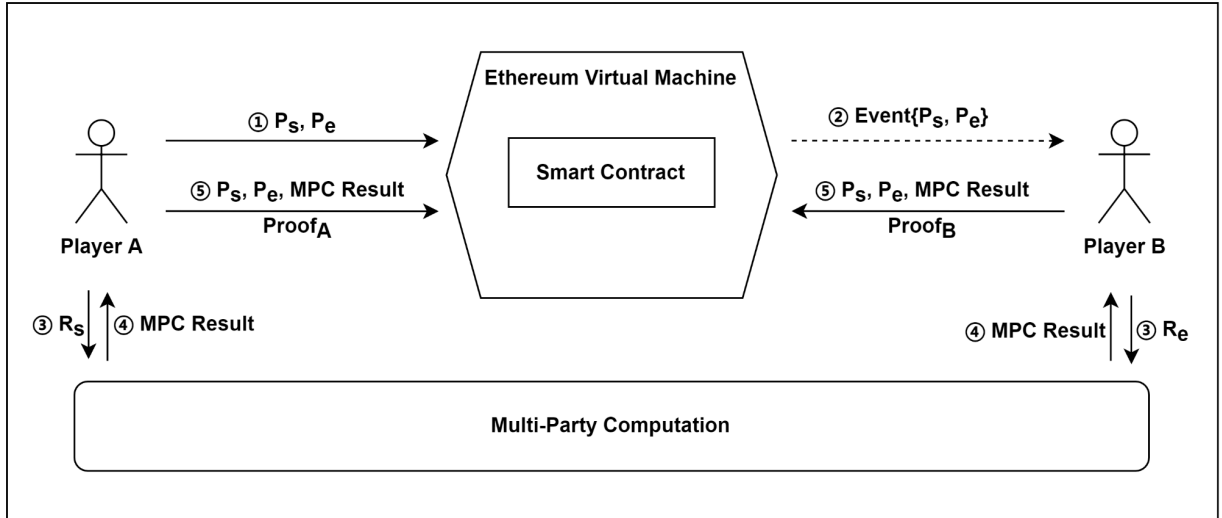


Figure 10. Shows players sending proofs to Ethereum during *Attack* state.

#### 4.3.5 Game State 5: End Game

Players take turns conducting the game actions of moving and attacking, transmitting between the *Move* and *Attack* states until one of the Flag is captured. The game program transmits to the *End Game* state the state diagram shown in *Figure 7*. The capturer is declared to be winner, and loser otherwise.

## Section 5. Application Findings & Analysis

### Section 5.1 Advantages of Decentralization

Through introducing decentralization into *ZK-Junqi*, it provides us with various advantages in comparison to a centralized game server. First, it does not rely on a trusted game server. Under the circumstance where a single point of failure occurs, works are redistributed to other active available nodes over the Ethereum network and hence it contributes to better user-experience. Second, deploying the game constraints over a public network advances the application with better trust between players. Since players' actions are validated openly by following the execution of the deploying, immutable smart contract, the validation process is reliable with high confidence. Third, the combining integration of zk-SNARK and MPC further improves the gameplay experience of *ZK-Junqi*. Without introducing a third role as central agent, the arbitration process of the rank comparison in *attack* can be done by players themselves. And the move actions validating with zk-SNAKE proofs are cryptographically secure and hence are reliable.

### Section 5.2 Challenges in Applied Decentralization

Along with our experiment, we find a few downsides of the decentralized technologies that we used in *ZK-Junqi*. In order to develop customized zk-SNARK proofs for *ZK-Junqi*, we use the circuit language `circom` and a javascript library `snarkjs`. The former allows us to prototype desired constraints in our zk-SNARK proofs, while the latter is used for translating circuit logic into executables that can be interpreted by a smart contract program (so that the proofs can be verified in a decentralized manner). Both technologies are relatively new and lacking documentation. And yet their compatibility cannot be guaranteed, although oftenly they are tightly used together in applied zk-SNARK. Their incompatibility has contributed to one of the big challenges we encountered during development.

### Section 5.3 Learnings & Future Works

A significant learning from developing *ZK-Junqi* is realizing the conflicts between academic theory and real-world problems in applied cryptography. When the MPC protocol is integrated as parts of optimization, it introduces a severe vulnerability to our application from a software engineering perspective. Since the secret inputs preparing for MPC computation are initiated and inputted by two players respectively, the fact that whether a player indeed input an X but Y cannot be verified. One the other hand, the zk-SNARK implementation in `snarkjs` cannot address this problem because it only allows a single-source of private inputs as one of the protocol/library limitations. Notice that the semi-honest security assumption that we make for *ZK-Junqi* is not enough for this particular use case of MPC. The reason is that confirming a player indeed input the real rank of his moving piece is critical in this game. Regardless of the



relaxing assumption upon the player's ability of learning other's input, our problem is still there. A dilemma that we are confronting here is either risking the vulnerability introduced by the current MPC implementation or modifying the validation process of the *attack* for the sake of security. As a big takeaway, real-world problems are complex, in need of comprehensive solutions in order to achieve real security, while the security syntax introduced in an academic environment oftenly is made under an ideal setting.

In seeking for a solution to our MPC challenge, we reached out to instructors and looked into the commitment scheme (X). Due to time issues, we could not further the project at this point. However, finding an appropriate implementation of MPC with commitment primitives would be valuable.

## Section 6. Conclusions

Using smart contracts in conjunction with zk-SNARK and MPC, we prototype and develop a decentralized chess game *ZK-Junqi*. Without introducing a central agent in *ZK-Junqi* for arbitration, the players themselves can jointly compute a critical function, rank comparison using the MPC protocol. As for the piece movement, all the players' moves are cryptographically verified and securely validated using the zk-SNARK protocol and Ethereum. Apart from the zk-SNARK and MPC, we also make use of a number of decentralized technologies such as smart contract, Truffle Suit and Metamask in realizing a decentralized *ZK-Junqi*. Although a decentralized game server has a number of advantages in comparison to a centralized game server, the load-balancing of work distribution and the reliability of smart contract execution, the development environment is not very developer-friendly with issues like incompatibility. A main takeaway of this project is realizing the conflicts between an academic theory and a real-world problem. Oftenly security syntax introduced in an academic environment is of an ideal setting. However, as seen in the MPC challenge that we encountered, real-world problems are more complex, and they need comprehensive solutions in order to achieve real security. As for our future work, integrating the current MPC protocol with commitment primitives will be valuable in furthering *ZK-Junqi*.

# Citations

- [1] Ethereum: <https://ethereum.org/en/>
- [2] Luzhanqi: <https://en.wikipedia.org/wiki/Luzhanqi>
- [3] GANACHE: <https://www.trufflesuite.com/docs/ganache/overview>
- [4] TRUFFLE: <https://www.trufflesuite.com/docs/truffle/overview>
- [5] Solidity: <https://docs.soliditylang.org/en/v0.8.10/introduction-to-smart-contracts.html#a-simple-smart-contract>
- [6] Snarkjs: <https://github.com/iden3/snarkjs>
- [7] Circom: <https://docs.circom.io/getting-started/installation/>
- [8] Web3.js: <https://web3js.readthedocs.io/en/v1.5.2/>
- [9] Commitment Scheme: [https://en.wikipedia.org/wiki/Commitment\\_scheme#Applications](https://en.wikipedia.org/wiki/Commitment_scheme#Applications)
- [10] Sun, X., Yu, F., Zhang, P., Sun, Z., Xie, W. and Peng, X., 2021. A Survey on Zero-Knowledge Proof in Blockchain. *IEEE Network*, 35(4), pp.198-205.
- [11] Courses.csail.mit.edu. 2021. [online] Available at: <<https://courses.csail.mit.edu/6.857/2020/projects/13-Gupta-Kaashoek-Wang-Zhao.pdf>> .
- [12] Ballesteros Rodríguez A. zk-SNARKs analysis and implementation on Ethereum[J].
- [14] Zero-knowledge proof: [https://en.wikipedia.org/wiki/Zero-knowledge\\_proof](https://en.wikipedia.org/wiki/Zero-knowledge_proof)
- [15] Multi-party computation: [https://en.wikipedia.org/wiki/Secure\\_multi-party\\_computation](https://en.wikipedia.org/wiki/Secure_multi-party_computation)
- [16] MPC Implementation: <https://github.com/markkurossi/mpc>

## Appendix A.

**Game Board:** The *Junqi* board is a mimic of a battlefield filled with featured barriers that affects players' moves. An example of *Junqi*'s game board is captured in *Figure 1* following with *Table X* summarizing associated game rules.

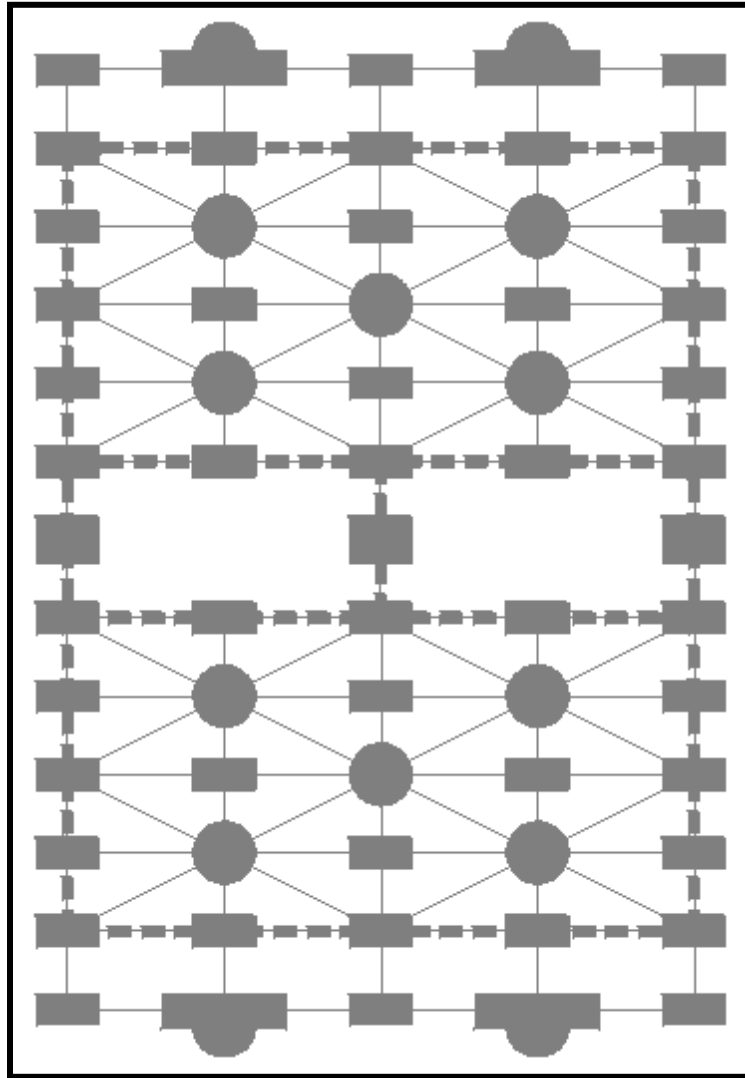



Figure 1. Shows the game board of *Junqi* following with featured barriers' explanation.

Table 1. Show the game constraints of each featured barrier on the *Junqi* game board.

Barriers	Appearance	Move Constraints
Soldier Station		Pieces can move on or off these spaces at will, and can be attacked and captured on them.

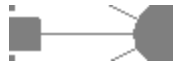

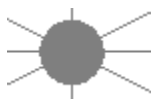


Roads		A piece can only travel one space across a road at any time.
Railroads		A piece can travel any number of spaces along a railroad in a straight line, as long as its path is not obstructed by another piece.
Campsite		A piece on a campsite cannot be attacked.
Mountains		No pieces can be placed on these two spaces.
Frontlines		No pieces can be placed on these spaces which are for crossover only.
Headquarters		Flag must be placed on one of these two spaces, and any piece that has been placed on or has entered headquarters can no longer move.



Figure 2. Shows the 12 game pieces owned by each player.

**Game Pieces:** Each player has 25 pieces, which are identical except for markings on one side. They are listed here in order of order, where any piece of a higher order may capture one of lower order (if the enemy is of the same order, both pieces will be removed from the board).

Table 2. Show the move's constraints according to each feature piece in *Junqi*.

Rank and Role followed with quantity	Move Constraints
Rank 1- Field Marshal (1)	If the opponent's Field Marshal is captured, then the location of the Flag piece has to be revealed but this depends on cases.

Rank 2 - General (1)	/
Rank 3 - Major General (2)	/
Rank 4 - Brigadier General (2)	/
Rank 5 - Colonel (2)	/
Rank 6 - Major (2)	/
Rank 7 - Captain (3)	/
Rank 8 - Lieutenant (3)	/
Rank 9 - Engineer (2)	Only pieces that can make turns when travelling along the Railroad. Engineers can also capture Landmines without being removed from the board.
Rank 10 - Bomb (2)	When in contact with any opponent piece, Bomb destroy both itself and the piece. They may capture the opponent's flag. Bombs cannot be placed on the front line (the first rank) during the initial set-up.
Rank 11 - Landmine (3)	Landmines are unmoveable but immune to any attack, and cause the destruction of attacking pieces (except when attacked by an Engineer or destroyed by a bomb).
Rank 12 - Flag (1)	Flag must be placed on one of the two Headquarters spaces on the sixth rank. It cannot move. Its capture brings the victory of the attacker and ends the game.