Spring 2023

Spartan Price Oracle: A Schelling-point Based Decentralized Pirce Oracle

Sihan He
San Jose State University

Spartan Price Oracle: A Schelling-point Based Decentralized Pirce Oracle

A Project

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Sihan He

May 2023

The Designated Project Committee Approves the Project Titled

Spartan Price Oracle: A Schelling-point Based Decentralized Pirce Oracle

by

Sihan He

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2023

Dr. Thomas Austin	Department of Computer Science
Dr. Mike Wu	Department of Computer Science
Dr. Justin Rietz	Department of Social Sciences

ABSTRACT

Spartan Price Oracle: A Schelling-point Based Decentralized Pirce Oracle

by Sihan He

Nakamoto's Bitcoin is the first decentralized digital cash system that utilizes a blockchain to manage transactions in its peer-to-peer network. The newer generation of blockchain systems, including Ethereum, extend their capabilities to support deployment of smart contracts within their peer-to-peer networks. However, smart contracts cannot acquire data from sources outside the blockchain since the blockchain network is isolated from the outside world. To obtain data from external sources, smart contracts must rely on Oracles, which are agents that bring data from the outside world to a blockchain network. However, guaranteeing that the oracle's off-chain nodes are trustworthy remains a challenge. A centralized oracle that relies on a single off-chain node creates a single point of failure. Therefore, a decentralized mechanism is necessary. One possible design for a decentralized oracle is to use a game mechanism that utilizes the Schelling-point theory to identify the correct data point among various data points reported by the oracle's off-chain nodes. In this paper, we introduce Spartan Price Oracle (SPO), a decentralized oracle designed to provide accurate price data. SPO utilizes the Schelling-point theory in its game mechanism to ensure the accuracy of the price data it provides. The mechanism design of SPO is based on SchellingCoin but with two significant improvements. Firstly, SPO uses Kernel Density Estimation to estimate the probability density function of data points that are reported by multiple off-chain nodes. This enables SPO to identify the accurate data by determining the mode of the probability density function. Secondly, SPO utilizes a redistributive economic incentive model that incorporates an appeal mechanism to increases the maximum reward for its off-chain nodes. This model has

been proven to raises the budget required for compromising off-chain nodes and helps in preventing potential attacks on the oracle.

TABLE OF CONTENTS

CHAPTER

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Nakamoto [1] invented Bitcoin as a decentralized digital cash system operating on a peer-to-peer network. Bitcoin employs a distributed database to manage transactions within the system, and updates to this database are only accepted if most peers in the network reach consensus on their validation. A distributed database managed through decentralized consensus, like Bitcoin, can also be referred to as a blockchain [2]. Digital cash systems that reply on blockchain technology for transaction management are known as cryptocurrencies [3]. The distributed database operating on a blockchain network is called the on-chain database.

Today, the applications of blockchain technology extend beyond cryptocurrencies. The newer generation of systems utilizing blockchain technology, including Ethereum [4], has the capability to execute smart contracts within their peer-to-peer networks [5]. Smart contracts are digital agreements written in code and enforced by blockchain technology when all conditions are met [3]. With the introduction of smart contracts, blockchain technology has expanded into fields other than cryptocurrencies, such as the Internet of Things(IoT) [6, 7], supply chain management [8], and clinical trials [9].

Smart contracts deployed on a blockchain cannot acquire data from sources outside the blockchain since the blockchain network is isolated from the outside world. To obtain data from external sources, smart contracts must rely on Oracles, which are agents that bring data from the outside world to a blockchain network. As discussed by Al-Breiki et al. [5], oracles work by having external actors bring data from external sources and report it to a blockchain through on-chain oracle smart contracts. The on-chain oracle smart contracts can be called oracle contracts, and the external actors can be called oracle nodes [10].

However, there is an issue with how oracles guarantee that their nodes are trustworthy. According to Al-Breiki et al. [5], a centralized oracle that relies on one oracle node to report data will not work because it creates a single point of failure if the oracle node dishonestly reports false data. Therefore, a decentralized oracle should be designed that does not rely on trusting in oracle nodes. One possible design of a decentralized oracle is to use a game mechanism that utilizes the Schelling-point theory to determine the correct data point among various data points that are reported by oracle nodes.

Alexander [11] has noted that Schelling points are the special points standing out among all the points that naturally become common targets that people will select to reach consensus without cooperating with each other in a coordination game. When voting for a choice that voters must reach an agreement on without cooperation in a voting game, Buterin [12] argues that the truth is the most obvious and powerful point among all possible choices for voters. SchellingCoin [12], is an iconic example of decentralized oracles that utilize the Schelling-point theory to obtain accurate data. It is a price oracle that employs a voting game to ensure accurate price data. As a decentralized oracle, SchellingCoin is comprised of an oracle contract and multiple oracle nodes. When the SchellingCoin oracle contract receives a data query, it initiates a voting game in which each oracle node must select a value from a set of distinct values to vote for by submitting the value to the oracle contract. Oracle nodes are not allowed to collaborate during the voting game. Once all oracle nodes have finished voting, the oracle contract sorts all the votes and determines the median vote as the most accurate value for the data query. The oracle contract then rewards the oracle nodes whose votes fall between the 25th and 75th percentile of the list of sorted votes. To be eligible for a reward, the oracle nodes must reach a consensus on submitting similar values to ensure that their votes fall between the 25th and 75th

percentile. In the absence of communication, the true data for the data query becomes the only Schelling point for oracle nodes to reach a consensus on. By utilizing the Schelling-point theory, SchellingCoin uses economic incentives to encourage oracle nodes to submit most accurate data, thus ensuring that data provided to other smart contracts is most accurate.

One issue with SchellingCoin is its method for determining the most accurate data. While SchellingCoin's approach works well when votes are normally distributed like a normal distribution, there may be cases where votes are not distributed normally. When a distribution of votes does not follow a normal distribution, the method of selecting the median as the most accurate value may not be effective, as the median vote may not reflect the value with the highest density of votes among all the available value choices.

For instance, figure 1 displays a histgram of votes where x-axis represents the value choices, and the y-axis represents the vote count for each value choice. The shape of the histogram clearly does not follow a normal distribution because there is a sudden increase in vote count at the value of 14. This surge in vote count can be a result of a bribery attack, where multiple oracle nodes have been bribed to vote for a false value. We can observe that the value with the highest density of votes is 10, indicating that most votes have been cast for the value of 10 and its neighboring value choices. However, the median vote in this case is actually the value of 11, which is not the value with the highest density of votes thus cannot be determined as the most accurate value according to the Schelling-point theory. Therefore, simply choosing the median vote as the most accurate value is not appropriate since it does not guarantee data correctness when a voting game is attacked. We need a better approach to identify the value choice with the highest density of votes.

Furthermore, SchellingCoin has room for improvement in its economic incentive

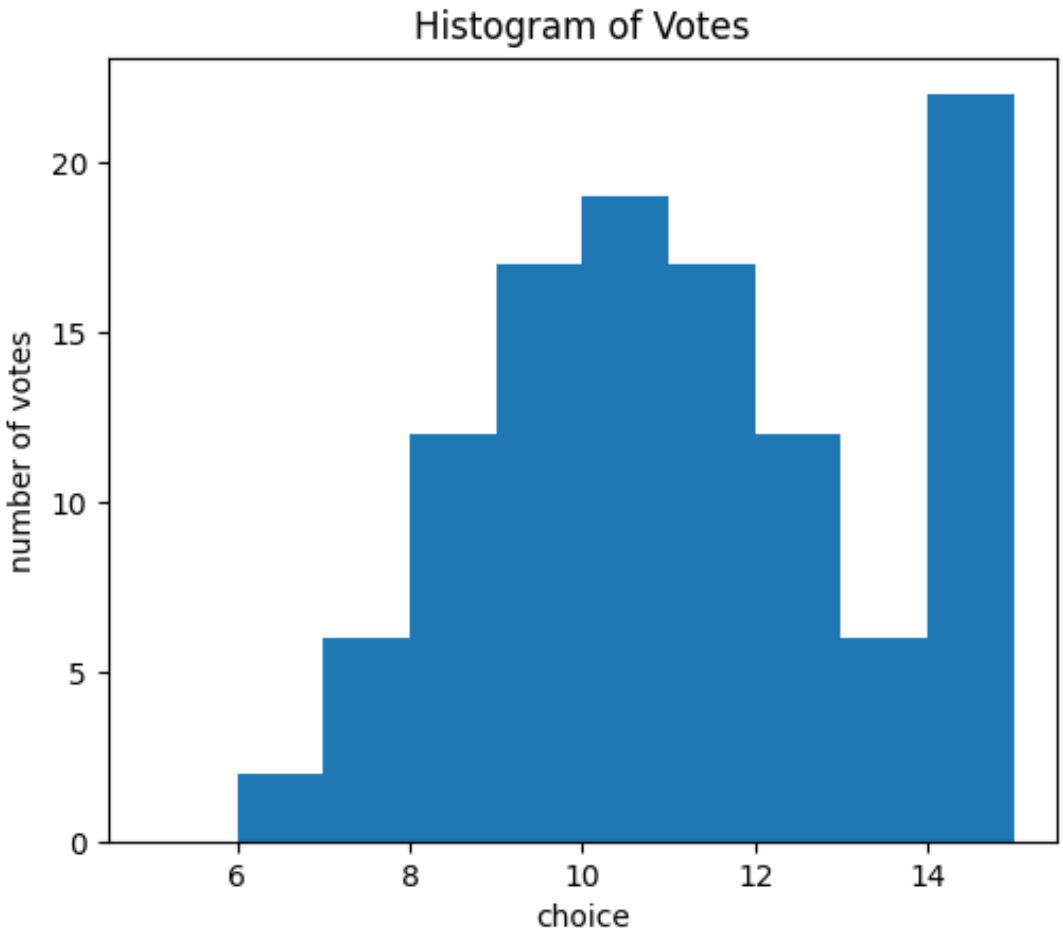

Figure 1: Plot of votes that are not distributed like a normal distribution

model. SchellingCoin uses a simple model that offers a fixed reward to each winning oracle node in a voting game but does not impose any penalties on the losing oracle nodes [12]. George and Lesaege [13] have shown that the budget required to bribe voters will only increase linearly with the number of voters to be bribed in a Schelling-point based game that utilizes an simple economic incentive model. An simple economic incentive model does not increase the cost of bribery and thus it is ineffective in preventing bribery attacks, such as the p+ϵ attack. To prevent such attacks, a more complex economic incentive model is required to increase the budget required by attackers to bribe oracle nodes.

In this paper, we present our contribution which is focused on enhancing Schelling-Coin's resistance against bribery attacks. Our approach involves the development of a price oracle called the Spartan Price Oracle (SPO) that can reliably provide accurate price data. SPO's mechanism design is based on that of SchellingCoin but with two significant improvements. The first improvement is implementing a new method that has more resistant to bribery attacks when determining the accurate data. The second improvement is implementing a different economic incentive model that increases the budget required by attackers to bribe oracle nodes significantly, thereby preventing the oracle from being attacked.

The first improvement uses the kernel density estimation (KDE) method to identify the correct data for a data query. Parzen [14] has noted that the KDE method is a non-parametric method to estimate the probability density function of a random variable. In the case of a price oralce, the random vairable represents the possible value choices that oracle nodes can vote for. By using the KDE method, we can estimate the probability density function to construct a probability density distribution of votes in a voting game. The mode of the probability density distribution is the value with the highest probability density of votes, which represents the most accurate value that most oracle nodes are likely to agree on. Based on the Schelling-point theory, the mode of the probability density distribution will be determined by SPO as the correct data, and the winning oracle nodes are the 50 percent of oracle nodes who have voted for value choices that are closest to the mode of the probability density distribution.

Another change is to use a more complex economic incentive model. We switched to an economic incentive model called a redistributive model, which have been discussed by George and Lesaege [13]. George and Lesaege [13] have shown that the redistributive model combined with the appeal mechanism increases the maximum reward for the winning oracle nodes and impose penalties to the losing oracle nodes. Under the

redistributive model, each oracle node must place a deposit before participating in a voting game. At the end of the game, the deposits are redistributed to the winning oracle nodes, while the losing oracle nodes will lose their deposits. The winning oracle nodes will not only be rewarded with a fixed prize but also a share of the deposits from the losing oracle nodes.

The appeal mechanism enables oracle nodes to create multiple voting rounds in a voting game. Each voting round has its own pool of deposits [13]. Each oracle node need to choose a voting round to place a deposit and vote for a value choice. All votes from any voting rounds in a voting game are still processed together to determine the most accurate value for a data query, but the deposits placed in a voting round will only be redistributed within the same round. Although SPO considers at least 50 percent of oracle nodes as winners in a voting game, the percentage of winning oracle nodes in a voting round can be less than 50 percent. When there are fewer winning oracle nodes but more losing oracle nodes in a voting round, winning oracle nodes can receive a larger reward by sharing the deposits from the losing oracle nodes in the round. The redistributive model, when combined with the appeal mechanism, can significantly increase the maximum possible reward for a voter, which increases budget required by attackers to bribe oracle nodes.

By implementing two improvements to the game mechanism, SPO will be able to reliably provide correct data to smart contracts on a blockchain network. Additionally, the new economic incentive model significantly increases the amount of funds attackers would need to bribe voters. These two improvements will enable SPO to improve its resistance against bribery attacks.

This paper is structured as follows. Section II briefly introduces blockchain, smart contracts, Schelling-point game theory, and some of the decentralized oracles that utilize Schelling-point game theory in their game mechanism. Section III explains

the game mechanism of SPO and defines the notations used in the paper. Section IV presents an implementation of SPO and its workflow. Section V discusses an analysis of SPO and result of experiments with the implementation of SPO. Section VI talks about future improvements to SPO. Section VII concludes this paper.

CHAPTER 2

BACKGROUND

2.1 Blockchain

A blockchain is a type of distributed ledger shared among nodes in a peer-to-peer network [3, 2]. It consists of blocks that store information about transactions and are linked together consecutively. Each block also stores the hash of its previous block to create a chain of dependencies. To add a new block to the blockchain, the nodes in the network must reach a consensus.

2.2 Blockchain Smart Contracts

Allam [2] has described a blockchain smart contract as an agreement written in code that can be deployed to a blockchain system. A smart contract enforces execution of a code if the corresponding conditions are satisfied. Al-Breiki at el. [5] has further elaborated that blockchain smart contracts enable various applications in blockchain systems. Examples of smart contracts used in fields other than cryptocurrencies are [8, 6, 9, 7].

2.3 Oracles

Oracles are agents who bring external data to smart contracts in blockchain systems [5]. Oracles are composed of two types of participants: oracle contracts and oracle nodes [10]. Oracle contracts are on-chain smart contracts, while oracle nodes are off-chain applications that provide data. When a smart contract requires data from off-chain sources in a blockchain system, it sends a data query to an oracle contract that can help it obtain the necessary data. After accepting the data query, the oracle contract broadcasts a data query event on the blockchain system network. Oracle nodes, which are off-chain applications, listen to the blockchain system network for data query events broadcasted by the oracle contract. After capturing the data query event, the oracle nodes attempt to extract data from external sources based on

the requirements of the data query event and then report the data to the blockchain system through the oracle contract. The oracle contract verifies the data provided by the oracle nodes and then provides the correct data to the smart contract that needs it.

2.4 Schelling Points

As discussed by Alexander [11], Schelling points (also called focal points) are specific points that stand out among all others and naturally become common targets for people to select if they need to reach consensus without cooperation. For instance, in a game with two players where each is asked to pick a number among all available choices without communication, and both players will be rewarded if they pick the same number. This game may seem challenging for players to earn a reward since the probability of both players picking the same number without communication is low. However, if there is a unique number that stands out among all the choices, it can become the obvious target for both players to pick to earn a reward.

2.5 Nash Equilibrium

Alexander [11] has defined a Nash equilibrium as the outcome of a game where each player follows their own strategy and will not change it, given what others are doing. For instance, in the prisoner's dilemma involving two suspects, A and B, both were arrested by the police and are locked in separate rooms with no means of communication. They must decide whether to confess or not, and their choices will result in different punishments, as shown in Figure 2.

If both suspects confess, each of them will receive a six-year prison term, while if both don't confess, each of them will get a three-year prison term. If suspect A confesses, but suspect B doesn't, suspect A will get a one-year prison term, and suspect B will get a nine-year prison term. Conversely, if suspect B confesses, but suspect A

	Suspect A confess	Suspect A does not confess
Suspect B confess	-6, -6	-1, -9
Suspect B does not confess	-9, -1	-3, -3

Figure 2: Outcome matrix for prisoners

doesn't, suspect B will get a one-year prison term, and suspect A will get a nine-year prison term. Although not confessing seems to be the best strategy for both suspects, the inability to communicate makes confessing the best option for both. If suspect B confesses, suspect A should also confess; otherwise, suspect A will receive a longer prison term. If suspect B doesn't confess, suspect A should still confess since suspect A will get a shorter prison term. The same holds for suspect B. Thus, regardless of what the other's decision is, both suspects would not regret to confess. This situation leads to a Nash equilibrium, where both suspects will not regret confessing, no matter the other's strategy.

2.6 P+ϵ Attack

Buterin [15] explains that a p+ϵ attack operates by persuading voters to vote for a false data in a voting game. The attacker promises to compensate the compromised voters if and only if they do not end up being winners of the voting game. The reward

matrix on the left in Figure 3 shows rewards paid to voters in a voting game without being attacked. It assumes that there are two choices of data: X and Y. As shown in the matrix, if a voter votes for a choice that most voters have voted for, the voter will be rewarded with a prize p. However, the reward matrix on the right in Figure 3 shows the reward matrix of a voting game after being attacked by a p+ϵ attack. In this matrix, a voter who votes for a choice that most voters have voted for will still be rewarded with a prize p. However, an attacker promises a reward of p plus ϵ to the voters who voted for Y if Y does not become the choice that most voters have voted for. This distorts the reward matrix, making the strategy of voting for Y a Nash Equilibrium because voters who vote for Y are guaranteed to get a reward, regardless of whether most voters voted for Y or not. If Y ends up becoming the choice that most voters have voted for, the attacker can potentially pay nothing, making the attack cost zero if it succeeds.

	Majoirty vote X	Majority vote Y
You vote X	p	0
You vote Y	0	p

	Majoirty vote X	Majoirty vote Y
You vote X	p	0
You vote Y	$p+\epsilon$	p

Figure 3: Reward matrix before and after being attacked

2.7 Schelling-point Based Decentralized Oracles

Decentralized oracles play an essential role in blockchain networks. Many smart contracts rely on decentralized oracles to bring the data they need from off-chain sources. Decentralized oracles commonly have a game mechanism that utilizes the Schelling-point theory to ensure data correctness. A game mechanism that utilizes

the Schelling-point theory uses economic incentives to reward players who reach consensus with most others on a data to report without communicating with each other. The game mechanism makes the strategy of reporting the true data a Nash equilibrium because the true data is the only Schelling point that most players are expected to report without communicating with each other. While utilizing the Schelling-point theory, different decentralized oracles can be designed in different ways. In this subsection, we will review some of the decentralized oracles that utilize the Schelling-point theory.

2.7.1 SchelllingCoin

SchellingCoin is a decentralized oracle invented by Buterin [12]. SchellingCoin has a game mechanism where a voter can gain a reward if the voter has voted for a value, which most voters have voted for, in a voting game. The game mechanism relies on the Schelling-point theory to find the true data since the true data is the only special value among all values that naturally becomes the common target for voters to vote for. The game mechanism finds the value that gets most votes as the true data in a game.

2.7.2 UMA Data Verification Mechanism

According to [16], the UMA Data Verification Mechanism (DVM) is a decentralized oracle that aimed to resolve disputes through a Schelling-point based mechanism. DVM uses the voting system like SchellingCoins to verify and provide the price of any assets. Voters need to hold the voting tokens in DVM. After voting, DVM collects the votes and calculates the mode, which is the value that gets most votes, as the verified price data. At least 50 percent of voters will be rewarded for the values that they have voted for are closest to the mode, and rest of the voters will be panelized. To prevent attacks on data, DVM enforces a mechanism that makes sure the cost

of corruption (CoC) to be always bigger than the profit from corruption (PfC). The mechanism will make sure bribe attacks are unprofitable. The CoC is measured by the minimum number of tokens that an attacker needs to prepare to control more than half of the voting power. The PfC represents the amount of wealth that can be won or lost in a smart contract, who is a registered user of UMA oracle, if data that it has queried from DVM oracle is manipulated. The CoC should be higher than PfC to make sure attackers cannot be profitable from a bribe attack. If CoC is approaching PfC, the DVM will charge fees from all registered smart contracts who are querying data from the DVM to buy back the voting tokens, which can drive the token price up to increase CoC.

2.7.3 Augur

Augur was invented by Peterson et al. [17]. Augur is both a decentralized oracle and a prediction market platform. Augur's native token is Reputation (REP). Augur allows anyone to create prediction markets for events and trade the shares of outcomes in the markets. A market creator who creates a prediction market needs to choose a designated reporter to report the outcome of the event. The creator also needs to specify a data feed where the reporter should bring the result after the event from. The creator also needs to post a bond that is paid in REP. The bond will be paid back to the creator if the designated reporter has reported an event outcome in time. If the reporter does not report in a period, any public reporters can compete to report. The first public reporter will be chosen to report the outcome. The bond will then be paid to the public reporter instead of returning to the creator. This mechanism incentivizes the market creator to choose a trustworthy reporter. After the outcome of the event has been reported, the market will move to a dispute round to resolve the disputing of the outcome. An outcome reported by a designated reporter, or a

public reporter is a tentative outcome since it is not verified yet. In a dispute round, any REP holders can participate by placing some REP as stakes on any outcomes other than the tentative outcome. If the amount of a dispute stake on an outcome reaches a required size, the outcome becomes the next tentative outcome, and the market moves to the next dispute round. If none of the dispute stakes reaches the required size, the tentative outcome is finalized to be the true outcome. The system will reward the participants who stake REP on the true outcome. The amount of reward that a participant will receive is proportional to the size of the stake that the participant has placed. Most of the stakes on the false outcomes will be redistributed to the participants who place stakes on the true outcome. Augur also relies on the Schelling-point theory to get the true outcome. It determines the outcome that gets the highest stake as the true outcome and rewards voters who put stakes on the true outcome. Again, the truth is the only Schelling point therefore it is most likely to be the common target that voters will be voting for without cooperations.

2.7.4 Razor

Razor is a decentralized oracle network developed by Huilgolkar [18]. Its native token is called ROZOR. There are two types of participants in Razor: clients and stakers. Razor divides time into separate epochs, processing the top J jobs in each one. Razor employs a verifiable random function (VRF) to select stakers to process jobs. Every chosen staker must process all J jobs and commit the results using the Merkle tree algorithm. Subsequently, Razor utilizes VRF to select jobs for each chosen staker to reveal the job results. The revealed job results can be disputed if their corresponding dispute bonds are filled within a specified period after the results are revealed. Finally, Razor applies VRF to sort stakers and selects the one at the top to propose a block. The staker who proposes the block uses the truth-by-consensus

mechanism to determine the truth from all revealed results for each job, and then take all true job result to create a block to propose. The block can also be disputed if deemed invalid. Razor rewards stakers who report the truth or propose valid blocks while penalizing dishonest ones. Razor identifies the job results that have reached consensus among selected stakers as the true job results. Stakers who report true job results will be rewarded. Since stakers cannot communicate with each other but must reach consensus, the truth naturally becomes the only Schelling point to report.

2.7.5 TruthCoin Blockchain

TruthCoin is a voting-based decentralized oracle and a prediction market that was invented by Sztorc [19]. It has two types of native tokens: CashCoins and VoteCoins. Users are allowed to create prediction markets and trade shares of outcomes on each market using CashCoins. The results of prediction markets are verified by voters who hold the VoteCoins. The outcome that receives most votes from VoteCoins holders in a prediction market is determined as the true outcome. TruthCoin uses economic incentives to encourage voters to participate in as many prediction markets as they can. A part of CashCoins collected from transactions are used to pay VoteCoins holders. The amount of CashCoins that a VoteCoins holder can receive is proportional to the number of VoteCoins they hold. TruthCoin uses singular value decomposition (SVD) to filter out outliers and rank voters by their influence in each market. VoteCoins will be redistributed to voters such that higher-ranked voters receive more VoteCoins, while lower-ranked voters receive fewer VoteCoins. Voters are ranked lower if their decisions do not reach consensus with most others' decisions or if they refuse to vote. When VoteCoins holders must reach consensus with most others in voting for a choice of outcomes, the true outcome, which is the only Schelling point, stands out to become the common target to vote for.

2.7.6 Witnet

Witnet was invented by Pedro et al. [20]. It is both a blockchain network and an oracle network. It has native tokens called Wit. Miners in Witnet are called witnesses. Witnesses are the computers that run software to automatically process clients' retrieve-attest-deliver (RAD) requests, which are data queries. Each witness has reputation points. A witness with more reputation points has a higher chance of being assigned to process a RAD request. A RAD request will be assigned to multiple witnesses at once. Witnet considers a witness's claim for a RAD request to be correct if it reaches consensus with most other witnesses' claims. A fee attached to a RAD task will be distributed as a reward to each witness who has a correct claim for the RAD request. A witness will also gain reputation points for having a correct claim for a RAD request. Others whose claims contradict the majority's claim will have their reputation points deducted. Witnet also uses the SVD algorithm to filter out outliers and detect collusion. In addition to processing RAD requests, witnesses can also participate in mining blocks. Witnet will select a subset of witnesses to create blocks using a deterministic algorithm. Witnesses with higher reputation points will be more likely to be assigned to mine blocks. The blocks need to be broadcast to the Witnet network and will be verified by the peers in the network before being accepted.

CHAPTER 3

SPARTAN PRICE ORACLE MECHANISM DESIGN

3.1 Game Mechanism Design

Spartan Price Oracle (SPO) is a price oracle that provides exchange rates between two assets. When the SPO contract receives a data query from another smart contract, it initiates a voting game that requires multiple oracle nodes to provide data through voting. The oracle nodes' task is to find the accurate exchange rate that the customer needs and vote for it in the voting game. As exchange rates are represented in numerical form, oracle nodes must select a value choice to vote for. Oracle nodes have an infinite number of choices due to the infinite numbers available.

A voting game consists of four phases. The first phase is the Commit phase. Typically, this phase has one commit round, but voters have the option to trigger the appeal mechanism and create multiple commit rounds within this phase. However, there is a fee charged by the SPO contract for triggering the appeal mechanism, and this fee doubles every time a new commit round is created in a voting game. The fee will eventually reach a point where voters can no longer profit from voting, thus preventing abuse of the appeal mechanism. Each commit round has its own deposit pool, and voters can choose which commit round to commit their votes. Before committing their votes, voters must place a deposit.

In this paper, we assume that there are totally N voters participating in a voting game. Each voter is only permitted to vote once in a voting game so there are N votes in total. We assume that there are M value choices for voters to select in the voting game.

We also assume that a voting game has K commit rounds. A commit round will be denoted by C_i where $i \in [1, K]$. Each commit round has its own deposit pool. The amount of assets in the deposit pool owned by C_i will be denoted by R_i.

To commit a vote, voters need to submit a hash of their vote in a commit round. Voters can use function (1) to generate a hash. In function (1), $hashVal$ denotes the hash that a voter should submit to the SPO contract for commiting a vote; `Keccak256` denotes the function of `Keccak256` secure hash algorithm; $Addr(\upsilon_{ij})$ denotes the account address of a voter; γ denotes the value choice that the voter wants to vote for; η denotes the nonce that voter picks randomly for using function (1).

$$hashVal = \texttt{Keccak256}(Addr(\upsilon_{ij}), \gamma, \eta) \tag{1}$$

The number of voters in different commit rounds can be different. The set of voters who have committed their votes in C_i will be denoted by V_i and the number of voters in V_i will be denoted by $|V_i|$. A voter in V_i is denoted by υ_{ij} where j is the id of the voter such that $j \in [1, |V_i|]$. As mentioned earlier, the total number of voters in the voting game denoted by N, thus we can tell that $N = \sum_{i=1}^{k} |V_i|$. Each voter in V_i needs to place a deposit, which is denoted by D, before committing a vote in C_i. The deposit will be added to R_i. Thus, the R_i can be calculated by $R_i = |V_i|D$.

The second phase is called the Reveal phase. During this phase, voters who have committed their vote in the previous phase are required to reveal their votes to the SPO contract. They need to disclose their votes(γ) and nonce(η), which is shown in function (1).

The third phase is called the Compute phase. During this phase, the SPO contract processes the votes that voters have revealed in the Reveal phase to determine the correct choice from among all possible value choices that have received votes. To accomplish this, the SPO contract uses the kernel density estimate (KDE) method to estimate the probability density function for the value choices that voters have voted for in a voting game. Then use the function to construct a probability density distribution of votes. Finally, SPO identifies the mode of the probability density

function as the correct choice, which will be returned to the smart contract that requires it.

To use the KDE method, we need to select both a kernel function and a bandwidth for it. There are various types of kernel functions, including Gaussian kernel, Epanechnikov kernel, and Biweight, which we can choose from. Different kernel functions can provide different levels of smoothness when estimating the probability density function. We selected the Epanechnikov kernel function and used Silverman's rule of thumb to select the bandwidth for the KDE function. The minimum bandwidth will be set to 2.

The final phase is the Withdraw phase. During this phase, the SPO contract will identify the winning voters. Approximately half of the voters who have selected the value choices that are most similar to the correct choice will be recognized as winning voters of the voting game. Winners will receive a prize as a reward and will be able to withdraw their deposit from the SPO contract. We separate the voters in V_i into two sets: the set of winning voters, which is denoted by W_i, and the set of losing voters, which is denoted by L_i. The number of winning voters in W_i will be denoted by $|W_i|$, and the number of losing voters in L_i will be denoted by $|L_i|$. Voters in L_i will not be rewarded and cannot withdraw their deposits back. On the other hand, Voters in W_i not only can withdraw their deposits, they will also equally share losing voters' deposits. Therefore, the reward that a winner can get will be equivalent to $\frac{C_i}{|W_i|}$. If $|W_i|$ is small but $|L_i|$ is large, the winning voters in W_i will be able to claim more rewards.

The number of winning voters and the number of losing voters in different commit rounds will likely be different. However, as required, at least 50 percent of voters should be determined as winning voters, thus the total number of winning voters must

be

$$\sum_{i=1}^{k} |W_i| \geq \frac{N}{2}$$

CHAPTER 4

IMPLEMENTATIOIN

To demonstrate the value of SPO, we have implemented it as an Ethereum smart contract. However, performing computations on the Ethereum network is not free, it costs gas, which is the unit of fees to do computation in Ethereum network [4]. If computational tasks have high complexity, the cost can be significant. Our implementation of the SPO contract utilizes the KDE method to estimate the probability density function of votes and determine the correct choice based on the mode of the probability density function. However, the KDE method we implemented has high complexity, which makes it expensive to run. Although our implementation is not efficient, we focus on improving SPO's security and data accuracy in this paper. We will leave the reduction of gas costs for future work.

4.1 Spartan Oracle Token

The Spartan Price Oracle (SPO) contract is implemented as an Ethereum smart contract. It has its own token, the Spartan Oracle Token (SOT). SOTs are fungible tokens that follows the ERC-20 token standard [21]. To generate SOT, users can deposit ether into the SPO contract, and to withdraw ether from the SPO contract, they can destroy their SOT tokens in the SPO contract. The exchange rate between SOT and ether depends on the number of total circulated SOT tokens in the market and the amount of ether that SPO holds. SPO holds ether assets as collateral, which supports the value of SOT. Therefore, the total value of circulated SOT tokens in the market is equivalent to the value of the Ether assets held by SPO. For instance, if SPO holds 1,000 ether, and the number of circulated SOT is 1 million, then the exchange rate between Ether and SOT token is 1 to 1000.

However, the exchange rate between SOT and ether is not fixed. SPO will continuously generate new SOT to reward voters who have won a voting game, which

can increase the number of circulated SOT tokens. The increase in the supply of circulated SOT can cause inflation, which can lower the value of each SOT token. Nonetheless, the inflation of SOT can incentivize SOT holders to participate in voting games where they can earn rewards by winning.

4.2 Participants

There are three types of participants in SPO: customers, voters, and operators. Customers are smart contracts that submit data queries to SPO to start voting games. The value choices that voters need to choose to vote for are unsigned integer numbers ranging from 0 to 2^{256}. Voters are participants who do research and vote for the value choice that they believe is the answer to the question. To vote, they must place a deposit of SOT. Each voter should only have one vote per voting game. The winners in a voting game can withdraw their deposits along with extra rewards from SPO. Operators are participants who keep the voting games running. They prevent the voting game from being suspended. Operators who have kept the voting game going will be rewarded with a small amount of SOT, and they do not need to place a deposit.

4.3 Game Structure

A voting game is initiated once a customer submits a price query to SPO using the `SubmitRequest()` method from the SPO smart contract. Upon receiving the price query, SPO will initiate a voting game to obtain an answer.

Figure 4 shows the implementation of the `SubmitRequest()` method. The `SubmitRequest()` method has four parameters:

- `asset1`, `asset2` --- The IDs of two assets inform SPO about the exchange rate that voters should look for. The SPO contract contains a list of asset IDs, which customers and voters can acquire from.
- `minimalVotesCount` --- the minimum number of voters that must have partici-

```
function SubmitRequest(uint asset1, uint asset2, uint minimalVotesCount, uint decimalPlaces) external returns (bool) {
    require(_phase == 0, "A request can only be submitted when there is no active voting game");
    require(msg.sender != address(0), "The sender cannot be address(0)");
    require(asset1 < _tokenNames.length, "The id of the source token is not found");
    require(asset2 < _tokenNames.length, "The id of the destination token is not found");

    // resets all states
    reset();

    // Stores customer address
    _customerAddr = msg.sender;

    // stores the requirement of minimal number of votes
    _minVotesCount = minimalVotesCount;

    // move to phase 1, set start time and duration for phase 1.
    _phase = 1;
    _phaseStartTime = block.timestamp;
    _phaseDuration = 600;

    // set round info
    _round = 1;
    _totalRounds = 1;

    // emit a data query event to Ethereum network
    emit DataQuery(asset1, asset2, decimalPlaces);
    return true;
}
```

Figure 4: Implementation of the SubmitRequest() method

pated in the voting game. If there are not enough voters, the game should be cancelled. The customer sets a minimum number of voters for security purposes. Without enough voters, attackers won't need to bribe many voters, making it easier for them to attack the voting game with a limited budget.

- `decimalPlaces` --- the accuracy of data that required by the customer. Equation (2) shows how `decimalPlaces` is used to define the accuracy for a number. In equation (2), the symbol $\mathcal{B}$ denotes the actual value of an exchange rate. The symbol $\mathcal{A}$ denotes the value choice that the voter should vote for. The symbol $\mathcal{P}$ denotes the value of the `decimalPlaces` parameter that is defined by a customer. For example, if an exchange rate is 6.954, and the decimalPlaces is set to be -2, then the number that voters should vote for is 695 since $695 = (\frac{6.954}{10^{-2}} + \frac{1}{2}) mod 1$. To cast their votes, each voter needs to utilize Equation (2) to determine $\mathcal{A}$, which represents the value choice that they should select.

$$\mathcal{A} = (\frac{\mathcal{B}}{10^{\mathcal{P}}} + \frac{1}{2}) mod 1 \tag{2}$$

As shown in Figure 4, when the `SubmitRequest()` method is called to initiate a data query, it sets the _phase variable to 1. This variable represents the current phase ID. As shown in Table 1, a voting game has four phases: the Commit phase, the Reveal phase, the Compute phase, and the Withdraw phase. Each phase has a id, and Since the Commit phase has an ID of 1, the `SubmitRequest()` method sets the current phase to the Commit phase and initializes other variables related to this phase. Finally, the method emits a `DataQuery()` event to notify voters about the query and allow them to participate in the voting game if they capture the event.

Table 1: Table of Phase IDs

ID	Phase
1	Commit
2	Reveal
3	Compute
4	Withdraw

Usually, a voting game progresses through its four phases in sequence. Each phase has a duration, and to move from the current phase to the next one, an operator needs to trigger the `MoveOnToNextPhase()` method in the SPO contract if the current phase times out. Any holder of SOT can become an operator. The SOT holder who successfully triggers the `MoveOnToNextPhase()` method first can move the voting game to the next phase and receive a small amount of SOT as a reward.

In some cases, a voting game can be cancelled if certain conditions are not satisfied. For example, if a game does not receive enough votes as required by the customer, the game will be cancelled, and the game will directly moved to the Withdraw phase by an operator, where every voter can take one's own deposit back.

4.4 Workflow

4.4.1 The Commit Phase

A voting game moves to the Commit phase once it starts. During this phase, a commit round begins, allowing voters to commit their votes within a specific time window. Each voter must place a deposit before committing their votes. To commit a vote, a voter must submit a hash generated using function (1), which is the hash function discussed in the previous chapter.

Because of the appeal mechanism, it is possible to have multiple commit rounds during the Commit phase in a voting game. Voters have the option to not commit their votes in the current commit round, and instead wait for the next commit round if there is one. The Commit phase ends when the last commit round times out. Once the Commit phase ends, one of the operators can trigger the `MoveOnToNextPhase()` method to move the voting game to the next phase.

Figure 5 depicts a sequence diagram illustrating the interactions among different roles in both the Commit Phase and Reveal Phase. Step (1) to Step (12) from the sequence diagram in Figure 5 represent the interactions between the SPO contract and the participants during the Commit phase of a voting game. In the diagram, there are six roles including the SPO smart contract (SPO_SC), a customer smart contract (Customer_SC), three voters (Voter1, Voter2, Voter3), and an operator (Operator). The diagram assumes that there is only one commit round during the commit phase. In the diagram, Customer_SC starts a voting game by triggering the `StartVotingGame()` method in SPO_SC. After the voting has started, the voting game will move into the Commit phase and starts a commit round. In step (2) to step (4), all three voters have placed a deposit to join the voting game. In step (5), Voter1 commits a vote by submitting a hash to SPO_SC. SPO_SC confirms that the hash is received by sending a True value back to Voter1 in step (6). In step

(7), Voter2 also commits a vote by submitting a hash and the hash is accepted by SPO_SC in step (8). Operator waits until the timeout of the last commit round. Since the current commit round will be the only commit round during the Commit phase, the Commit phase will also be timed out when the current commit round is timed out. In step (9), Operator checks whether the commit phase is timed out by asking SPO_SC. SPO_SC replies to Operator that the Commit Phase is timed out in step (10). Therefore, Operator calls the `MoveOnToNextPhase()` method to move the voting game into next phase, which is the Reveal phase, in step (11). The method is successfully executed, thus SPO_SC sends a True value to Operator for confirmation. Now the voting game is in the Reveal phase, and SPO_SC starts a reveal round. However, voter3 tries to commit a vote during the Reveal Phase in step (13). It is obvious that SPO_SC must decline Voter3's commitment. Therefore, in step (14), SPO_SC sends a False value to Voter3 for notifying the decline.

If a game does not receive enough votes during the Commit phase, it will be cancelled. The operator can then move the game directly to the Withdraw phase after the Commit phase has timed out. During this phase, every voter can withdraw their deposit since there are no winners due to the cancellation of the game.

4.4.1.1 The Appeal Mechanism

During the Commit phase, voters may activate the appeal mechanism by calling the `Appeal()` method in the SPO contract. There is no hard limit on the number of times the appeal mechanism can be triggered, but it is not free. Voters must pay a fee each time they activate the mechanism. The fee will double every time the mechanism is triggered within a voting game. This increase in the fee is intended to prevent voters from abusing the mechanism to create excessive commit rounds, as the cost of activating it will quickly become expensive.

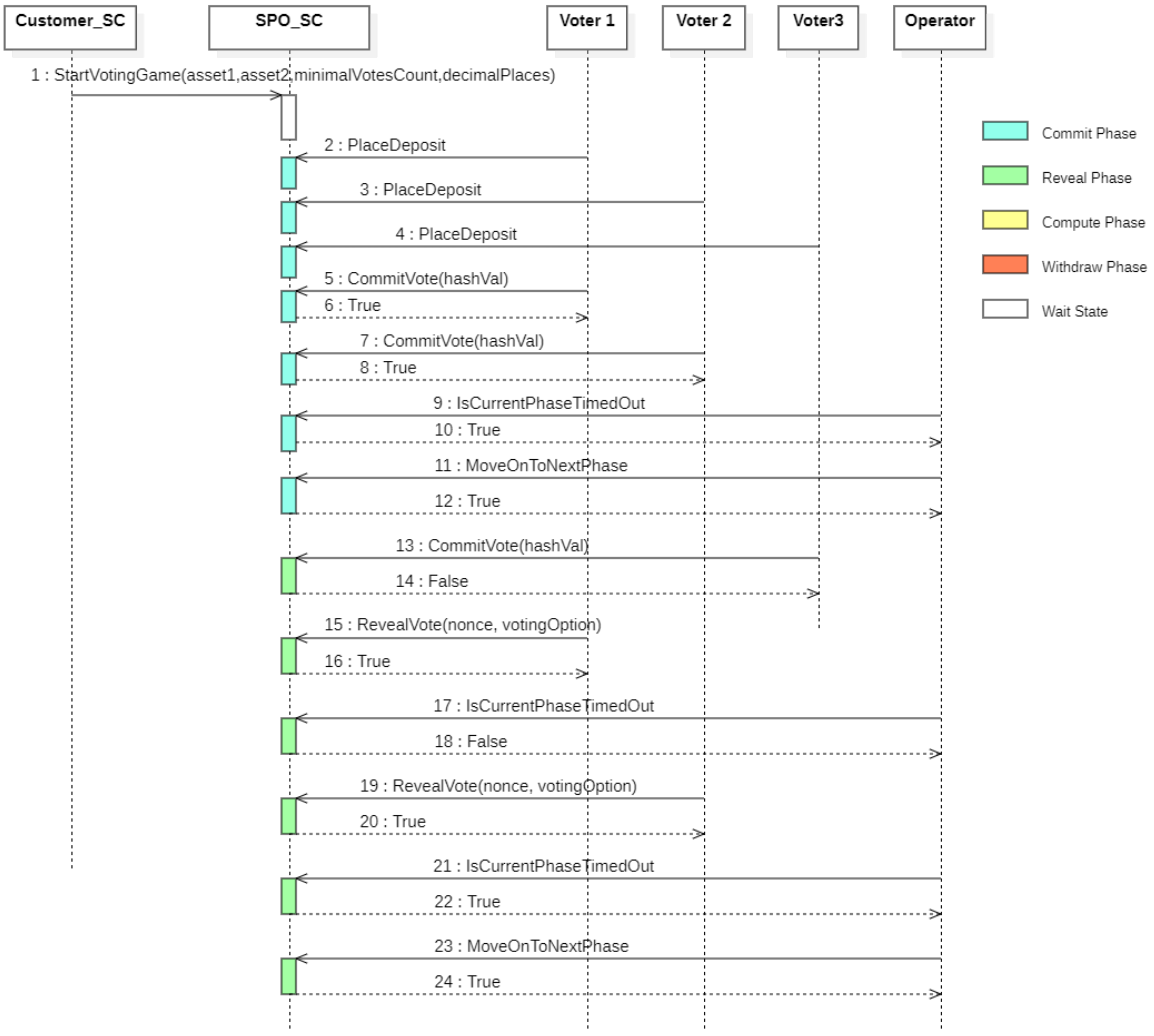

Figure 5: The sequence diagram for the Commit phase and the Reveal phase

Figure 6 presents a sequence diagram that shows a sequence of triggering the appeal mechanism by voters. In the diagram, the voting game is in the Commit phase which voters can trigger the appeal mechanism. The diagram has three roles: SPO smart contract (SPO_SC), and two voters (voter1, voter2). In step (1) and step (2), voter2 just commits a vote regularly. In step (3), voter1 uses the GetCurrentRoundId() method in SPO_SC to check the id of current commit round. SPO_SC replies to voter1 with an integer 2 to tell voter1 that the id of the current commit round is 2 in

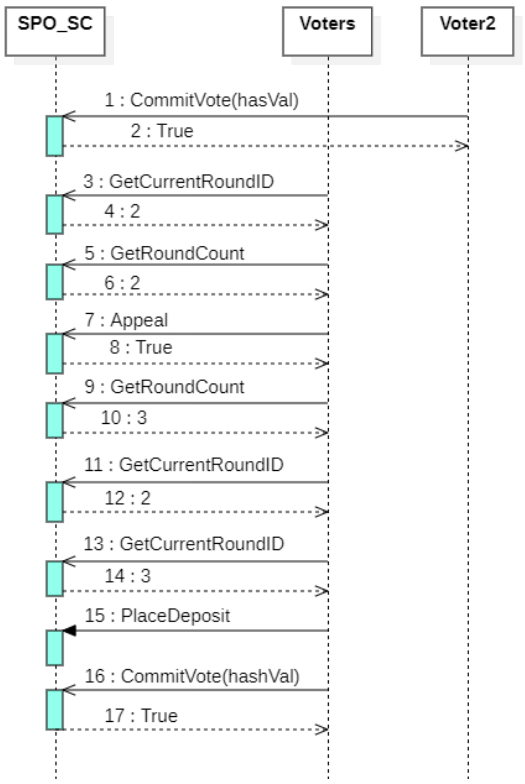

Figure 6: The sequence diagram for SPO's appeal mechanism

step (4). In step (5), voter1 uses the GetRoundCount() method in SPO_SC to check the total number of commit rounds for the current phase. SPO_SC replies to voter1 with an integer 2 to tell voter1 that there are 2 commit rounds in total in in step (6). Since the total number of commit rounds is 2 and the ID of the current commit round is also 2, it means that the current commit round is the last commit round for the current phase. In step (7), voter1 decides to use the appeal mechanism to create an additional commit round by calling the appeal() method in SPO_SC. The appeal mechanism is successfully triggered as shown in step (8) and a new commit round is created thus the total number of commit rounds in the current phase is now 3 as presented in step (10). In step (11), voter1 checks the id of the current commit round. The id of the current commit round is still 2 as shown in step (12) so voter1

decides to wait for a bit. In step (13), voter1 checks the id of the current commit round again. This time, the id of the current commit round is 3 as shown in step (14). Voter1 wants to commit a vote in the commit round 3 so voter1 places a deposit in the commit round 3 first in step (15). Then, Voter1 commits a vote in the commit round 3 in step (16). The vote commitment is accepted by SPO_SC in step (17).

4.5 The Reveal Phase

During the Reveal phase, voters are required to reveal the votes they previously committed. They must reveal their votes (γ) and nonce (η), which they used to generate their hashes, to the SPO contract. This allows the SPO contract to use the hashes that voter have submitted in the previous phase for verifying the revealed votes. Failed to reveal the correct votes will result in voters losing the voting game and not being able to retrieve their deposits. All γ that voters reveal will be stored in on-chain storage and later processed by the SPO contract to determine the outcome of the voting game.

Starting from step (13), the sequence diagram in Figure 5 shows the interactions between voters and SPO_SC in the Reveal phase. In step (13), voter3 tries to reveal a vote to SPO_SC but voter3 is rejected by SPO_SC in step (14) because voters3 has not commit the vote in the previous phase. In step (15), voter1 reveals the vote that voter1 has committed in previous phase to SPO_SC. Voter1's revealing is accepted in step (16). In step (17), Operator checks whether the current phase has timed out. SPO_SC replies to Operator that the current phase is not timed out yet in Step (18). Therefore, in step (19), voter2 reveals the vote that has been committed in earlier phase to SPO_SC and the reveal is successfully accepted by SPO_SC in step (20). In step (21), Operator checks whether the current phase has timed out again. SPO_SC replay to Operator that the current phase is timed out already in Step (22),

so Operator trigger the voting game to move to the next phase, which is the Compute phase, in step (23). The voting game is successfully moved to the Reveal phase in step (24).

4.6 The Compute Phase

During the Compute phase, the customer who made the data query must call the `FindResult()` method in the SPO contract to process the votes. The `FindResult()` method determines the mode with the probability density function estimated by the KDE method. In this phase, voters will not interact with the SPO contract. However, operators must advance the voting game to the next phase once the `FindResult()` method completes processing or the Compute phase times out.

The `FindResult()` method in the SPO smart contract processes the votes in two steps:

1. Sort votes and drop outliers.

2. Use KDE function to find the correct choice.

4.6.1 Delete Outliers

The first step of the `FindResult()` method is to sort the value choice that have received at least one vote. SPO uses the quicksort algorithm to perform the sorting. Next, the method removes the outliers, which are the edge value choices that barely receive votes and are distant from most other choices that have received votes. Since outliers are unlikely to be relevant, they can be removed. SPO uses the common 1.5 * Interquartile Range (IQR) method to identify the outliers. The voters who have voted for outliers will lose their deposits because they have no chance of winning the voting game.

4.6.2 Finding the Winning Choice

After removing the outliers, the `FindResult()` method employs the KDE method to estimate the probability density function of votes. The mode of the probability density function is the value choice has the highest possibility to receive votes from oracle nodes. Therefore the SPO contract declares it as the correct choice.

If the probability density function has multiple modes, The SPO contract will select the one that is closest to the median of the probability density function as the correct choice. This choice should have the minimal deviations from the other votes.

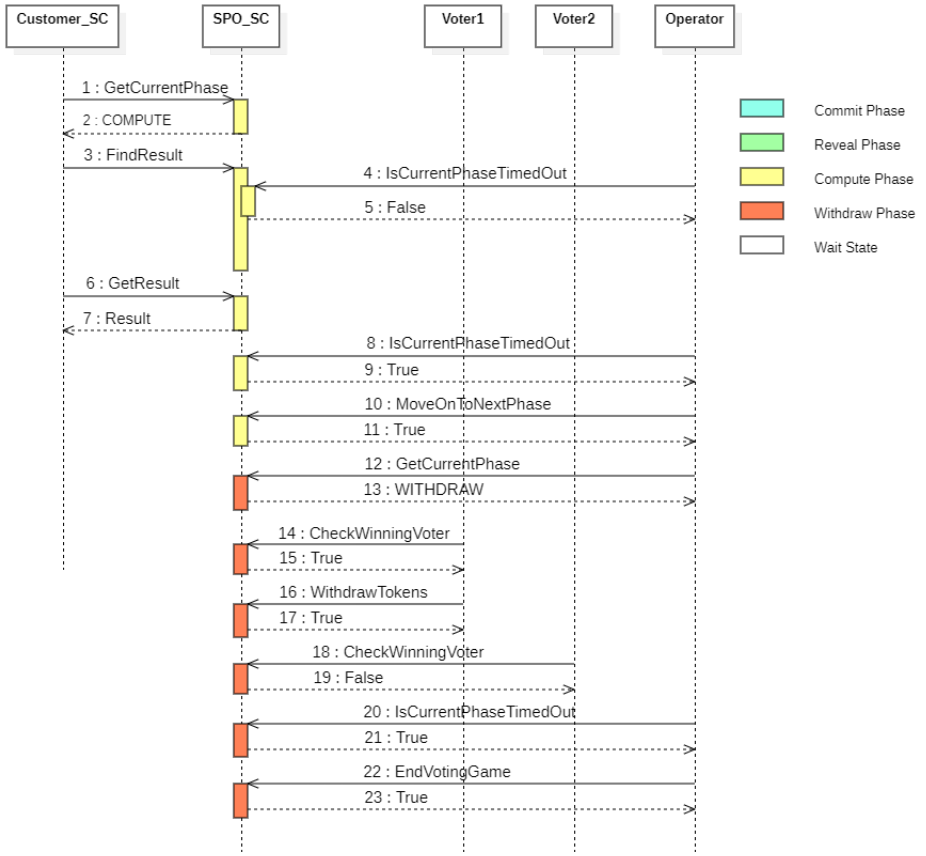

Figure 7: The sequence diagram for the Compute phase and the Withdraw phase

Figure 7 presents the sequence diagram of the Compute phase and the Withdraw phase. There are 5 roles including a customer smart contract (Customer_SC), a

SPO smart contract (SPO_SC), 2 voters (voter1 and voter2), and an Operator. In step (1), Customer_SC checks the current phase of the voting game by calling the `GetCurrentPhase()` method in SPO_SC. SPO_SC replies to Customer_SC that the current phase is the Compute phase. In step (3), Customer_SC calls the `FindResult()` method in SPO_SC to process the votes and find the result. If Operator checks whether the current phase has timed out while the `FindResult()` method is processing as shown in step (4), SPO_SC will reply to Operator with a False value as shown in step (5). In step (6), Customer_SC requests to get the result from SPO_SC after the `FindResult()` method has finished processing. SPO_SC returns the result to Customer_SC in step (7). In step (8), Operator checks whether the current phase is timed out again. Since the result has been found, SPO_SC reply to Operator with a True value to tell Operator that the Phase has timed out and the voting game is ready to move to the next phase. In step (10), Operator requests to moves the voting game into the next phase. And the voting game has been successfully moved to the Withdraw phase in step (11).

If Customer_SC does not call the `FindResult()` method during the Compute phase, the voting game will be canceled. In this case, the operator can move the game to the Withdraw phase once the Compute phase has timed out. During the Withdraw phase, every voter can withdraw their deposit since there are no winners due to the cancellation of the game.

4.7 Withdraw Phase

During the Withdraw phase, voters can withdraw their deposits and extra rewards if they have voted for the correct choice. If less than fifty percent of voters have voted for the correct choice, voters who have voted for the value choices that are closest to the correct choice can also withdraw their deposits and extra rewards from the SPO

contract. SPO ensures that at least 50 percent of voters are identified as winners of a voting game.

To check if they are winners of a voting game, voters can use the `CheckWinningVoter()` method in the SPO contract during the Withdraw phase. To withdraw their deposits and rewards, voters can use the `WithdrawTokens()` method in the SPO contract during the same phase.

From Fig. 7, the steps between step (12) and step (23) shows the sequence of interactions between an operator (Operator), two voters (voter1 and voter2), and SPO smart contract (SPO_SC) in a voting game. In step (12), Operator requests to check the current phase of the voting game to SPO_SC, and SPO_SC replies to Operator that the current phase is the Withdraw phase. In step (14), voter1 asks SPO_SC whether voter1 is a winning voter. SPO_SC replies voter1 with a True value in step (15), indicating that the voter1 is a winning voter. Therefore, in step (16), voter1 requests to withdraw tokens, which includes voter1's deposit and reward, from SPO_SC. And SPO_SC approves voter1's request in step (17). In step (18), voter2 also requests to check whether voter2 is a winning voter. And unfortunately, voter2 is not a winning voter so SPO_SC replies voter2 with a False value in step (19). In step (20), Operator checks if the current phase has timed out. SPO_SC confirms that the current phase has timed out in step (21). Thus, in step (22), Operator requests to end the voting game by calling the `EndVotingGame()` method in SPO_SC. And SPO_SC ends the voting game in step (23).

CHAPTER 5

ANALYSIS

5.1 Performance Analysis

George and Lesaege [13] proved that a Schelling-point game incorporating a redistributive model and an appeal mechanism can quadratically increase the budget required for attackers to bribe voters compared to a simple Schelling-point game. SPO has implemented the redistributive economic incentive model and the appeal mechanism in its mechanism, thus it is costly to bribe voters in SPO.

However, a concern is the number of voters that attackers need to bribe to manipulate a game result in a oracle. For an oracle that answers Boolean questions, an attacker would need to prepare a budget sufficient to bribe 50 percent of voters since there are only two choices. But the situation is getting more complex for price oracles because the data that price oracles provide are numeric values, and numeric values can be any numbers. Due to the uncertainty in measurements, values provided by various off-chain sources may slightly vary in accuracy, despite all of the values being accurate. It is challenging for most voters to agree on a single choice when there are mulitple valid choices that voters can select from when casting their votes. SPO uses a density based method to determine the most accurate value among all value choices that have gotten votes. In SPO, if honest votes in a voting game are not concentrated in one or a few value choices but are distributed evenly among multiple choices, the voting game will not have a accurate value with a high density of votes. When the density of votes at the most accurate value is not high enough, attackers only need to bribe less voters to vote for a false value, making it the value with the highest density of votes. As the result, the false value will be determined by SPO as the most accurate value, and SPO will fail to prevent the attacks.

To find out how distribution of honest votes can influence the oralce's resistance

against bribery attacks, we made five experiments. Each experiment simluated a voting game that was targeted by a attack. In our experiments, all honest votes were distributed normally like a normal distribution. To simulate scenarios in which honest votes are concentrated in one or a few value choices, we decreased the standard deviation of honest votes. Conversely, to simulate scenarios where honest votes are evenly spread across multiple choices, we increased the standard deviation of honest votes. To simulate a bribery attack, we made the assumption that all compromised voters would solely vote for a false value choice as appointed by the attackers.

Table 2: Specs of voting games in experiments

Experiment ID	1	2	3	4	5
Standard deviation of honest votes	2	2	1	1	1
Count of honest votes	200	200	200	200	200
Count of dishonest votes	80	100	100	140	180
Desired accurate value	200	200	200	200	200
False value	205	205	205	205	205

Table 2 shows the specs of all voting games that we created in our experiments. The number of honest votes in each game was identical, which was 200. The distribution of honest votes in each game was centered around the value choice of 200. However, the standard deviation of honest votes in different game could be different. In first two experiments, the standard deviation of honest votes in each voting game was set to 2. In last three experiments, the standard deviation of honest votes in each voting game was set to 1, which was lower compared to first two experiments. Also, each game was targeted by a different bribery attack. As shwon in Table 2, the number of dishonest votes would be different in different games. By doing so we could observe how standard deviation of honest votes could influence SPO on preventing different bribery attacks. We also conducted comparisons between SPO

and SchellingCoin to observe thier rections to the attacks and to determine their abilities to prevent such attacks.

The voting game in experiment 1 had 200 honest votes and 80 dishonest votes. Figure 8 shows the histogram of votes in the voting game. In the voting game, honest votes were distributed in a normal distribution that was centered around the value of 200, indicating that majority of honest voters believed that 200 was the most accurate value. The standard deviation of honest votes was 2, which indicates that honest votes were distributed evenly among multiple value choices. All 80 dishonest votes, on the other hand, were voted for the value of 205.

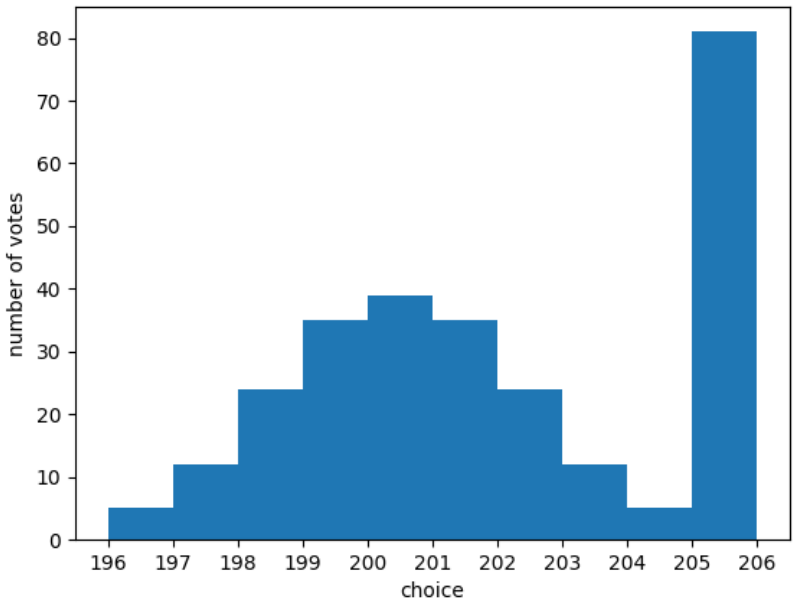

Figure 8: The histogram of votes in experiment 1.

In Experiment 1, we tested SchellingCoin first to observe the value that it would delilver. SchellingCoin determined the value of 201 as the most accurate value because the median vote was the value of 201. However, the most accurate value that the majoirty of honest voters had agreed on is 200. Therefore SchellingCoin

failed to prevent the bribery attack in this case since the resulting value produced by SchellingCoin had deviated from the expected value.

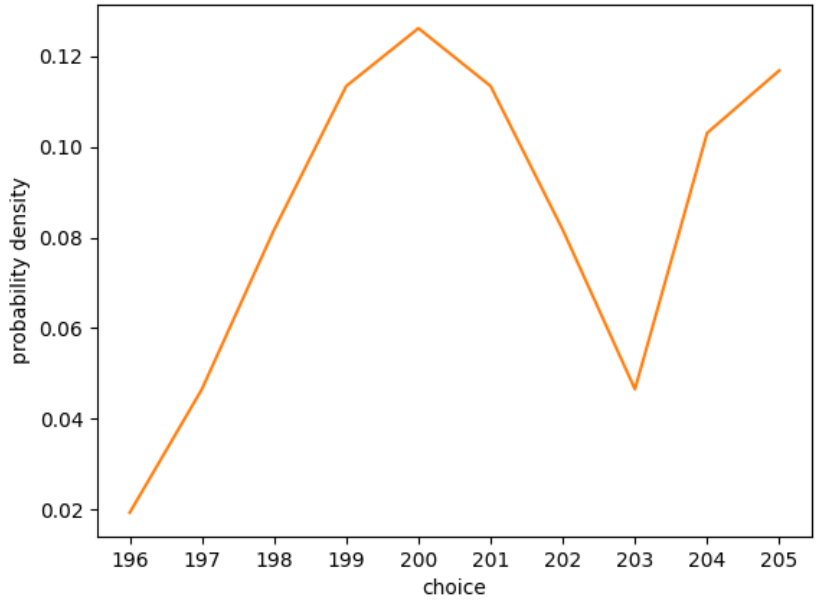

Figure 9: The probability density distribution of votes in experiment 1.

Then we tested SPO. SPO used the Kernel Density Estimation (KDE) method to estimate the probability density function of votes, which allowed SPO to construct the probability density distribution that is displayed in Figure 9. Since the mode of the distribution was at the value of 200, SPO would determine 200 as the most accurate value. The resulting value that SPO produced did not deviate from the expected value, therefore SPO successfully prevented the bribery attack in this case.

In experiment 2, we increased the number of dishonest votes to 100 while keeping other specifications unchanged from experiment 1. The histogram of votes are displayed in Figure 10. In this experiment, SchellingCoin again delivered the resulting value of 201, which deviated from the expected value of 200. On the other hand, SPO also failed to produce the expected value. Figure 11 is the plot of the probability density

distribution that was produced by SPO using the KDE method. As we can see in Figure 11, the mode of the distribution is not at the value of 200, but instead at the value of 205. As a result, SPO would determine the value of 205 as the most accurate value, which deviated further from the expected value. Compared to SchellingCoin, SPO actually performed worse in this case after it failed to prevent the bribery attack in this experiment.

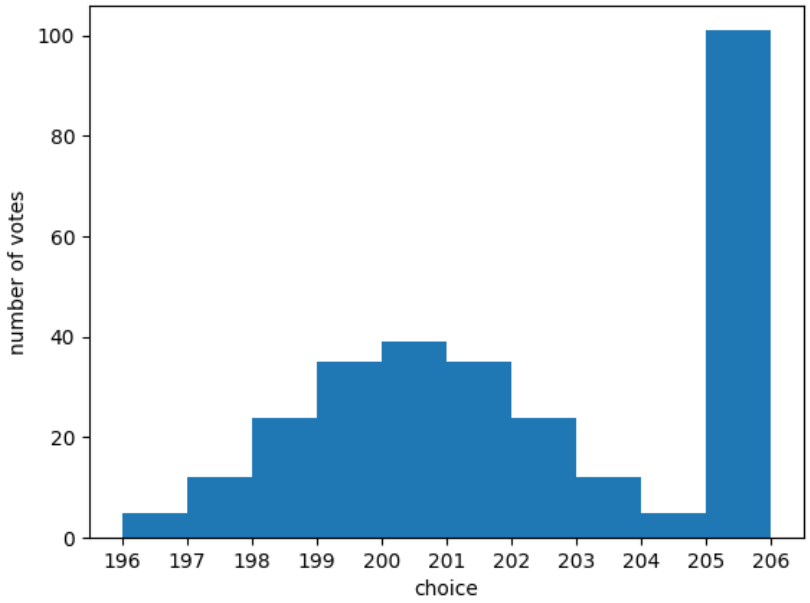

Figure 10: The histogram of votes in experiment 2.

Both SchellingCoina and SPO failed to prevent the attack in experiment 2. In experiment 3, we decreased the standard deviation of honest votes, making honest votes more dense at the value of 200. Again, the value of 200 is the value that we expected to get from both SchellingCoin and SPO. Other specifications of the voting game in experiment 3 remained the same compared to the voting game in experiment 2.

Figure 12 shows the histogram of votes for the voting game in experiment 3.

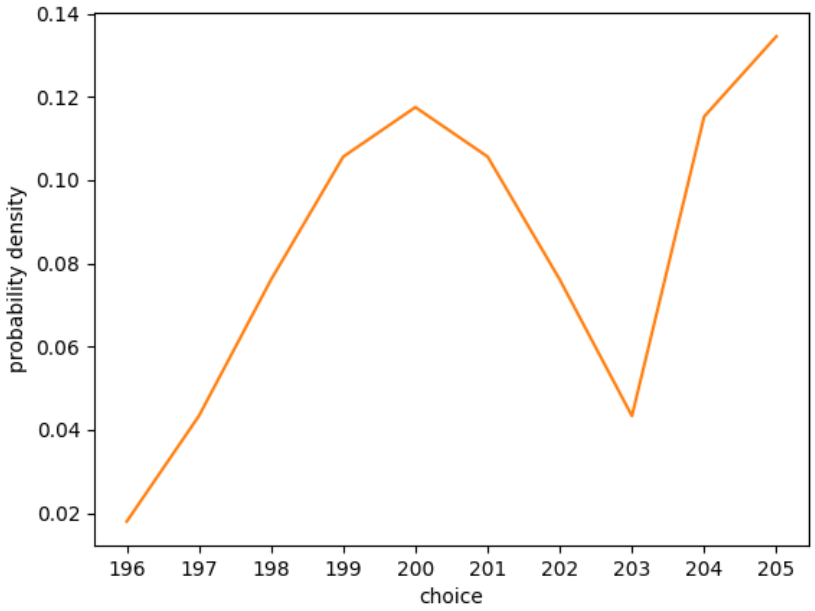

Figure 11: The probability density distribution of votes in experiment 2.

The median vote, again, is 201. So SchellingCoin would still determine the value of 201 as the most accurate value, which deviated from the expected value of 200. Therefore, SchellingCoin failed to prevent the attack in this case even though the standard deviatioin of honest votes was lower compared to experiment 2.

On the other hand, SPO would able to prevent the attack in this experiment. Figure 13 displays the probability density distribution of votes that SPO created using the KDE method in experiment 3. The mode of the probability density distribution is clearly at the value of 200, which is the value that we expected it to deliver. We can see that SPO's resistance against the bribery attack had increased due to lowering the standard deviation of honest votes. SPO's KDE method is density based, which highly replies on the density of honest votes at the accurate value. To improve SPO' resistance against bribery attacks, it is also important to make sure that honest votes does not spread widely among multiple choices.

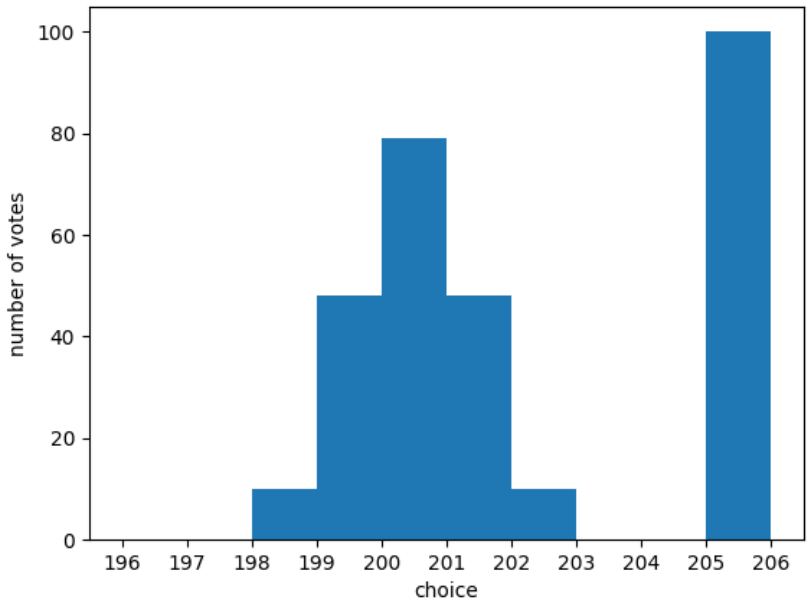

Figure 12: The histogram of votes in experiment 3.

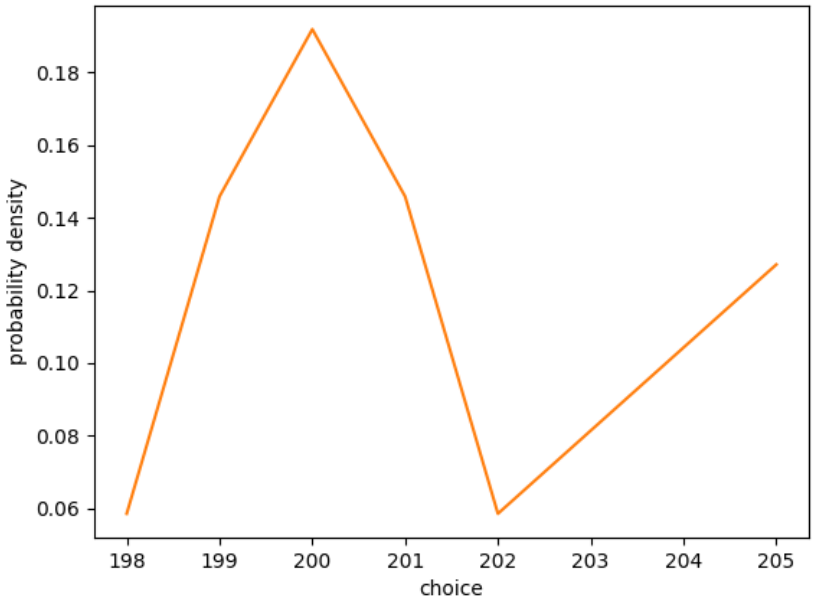

Figure 13: The probability density distribution of votes in experiment 3.

In experiment 4, we further increased the number of dishonest votes in the voting game from 100 to 140. Other specifications of the voting game remained identical compared to the voting game in Experiment 3. Figure 14 shows the histogram of votes in experiment 4. Again, SchellingCoin in experiment 4 failed to deliver the accurate value we expected. So SchellingCoin cannot prevent the bribery attack in this case. However, SPO could still prevent the attack and deliver the accurate value that we expected in this experiment. Figure 15 is the probability density distribution of votes that SPO would produce using the KDE method. The mode of the probability density distribution was at 200, which did not deviate from the expected value. In Experiment 4, the voting game has 340 votes in total, and the number of dishonest votes is roughly 40 percent of the votes in the voting game, which is tough for SPO to resist. However, SPO still managed to prevent the attack, showing that low standard deviation of honest votes can significantly improve SPO's resistance against bribery attacks.

In experiment 5, we further increased the number of dishonest votes to 180 and other specifications remianed the same compared to the experiment 4. Figure 16 shows the histgram of the votes in experiment 4. In this case, both SchellingCoin and SPO failed to prevent attacks. SchellingCoin determined the value of 202 as the most accurate value, which deviated from the expected value. SPO, on the other hand, delivered a value of 205 which further deviates from the expected value of 200. As shown in Figure 17, the false value, which is 205, is the value with the highest vote density. The percentage of dishonest votes is almost 50 percent of the total votes, therefore it is not surprise that SPO failed to prevent the attack in experiment 5. Again, when SPO fails to prevent an bribery attack, it performs worse when compared to SchellingCoin.

To prevent attacks, customers should consider sacrificing some accuracy to reduce

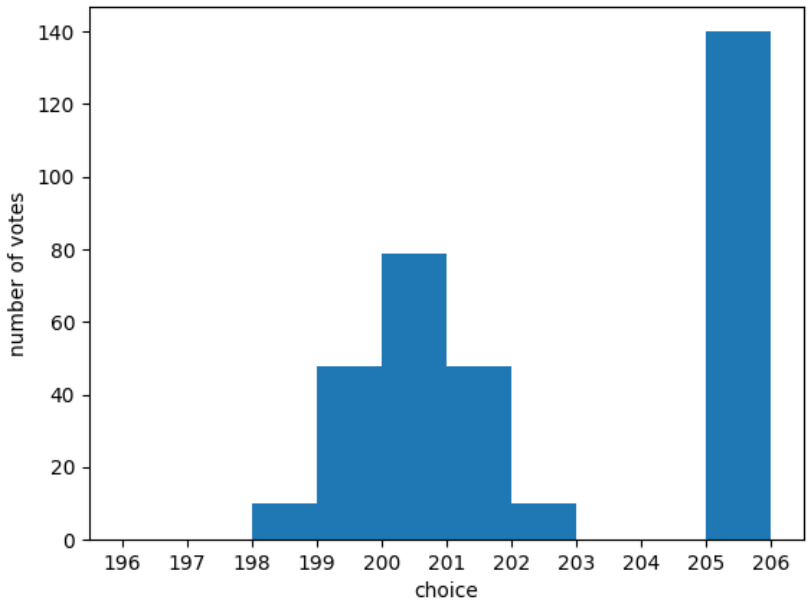

Figure 14: The histogram of votes in experiment 4.

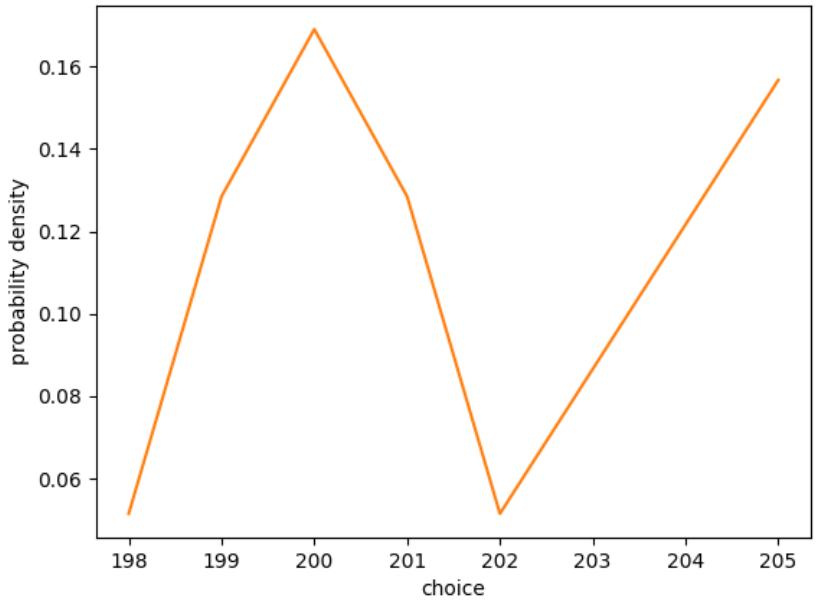

Figure 15: The probability density distribution of votes in experiment 4.

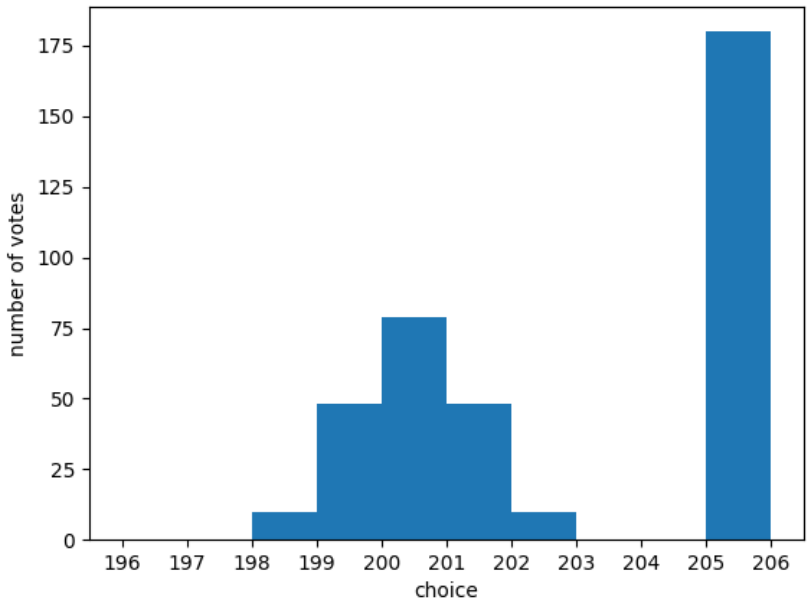

Figure 16: The histogram of votes in experiment 5.

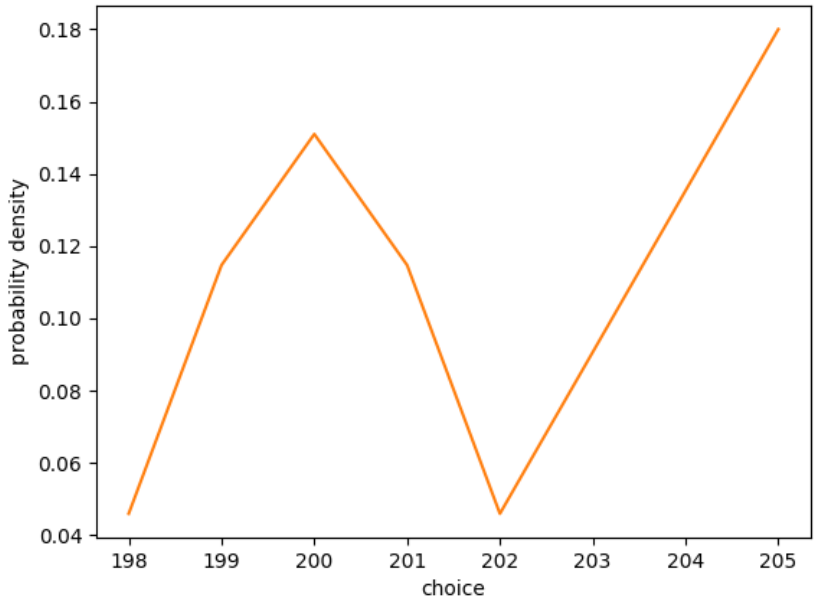

Figure 17: The probability density distribution of votes in experiment 5.

the valid choices that honest voters can select from when casting thier votes. For example, suppose there are four sources providing data with four different values. Source 1 claims the exchange rate is 112.53, source 2 claims it is 112.51, source 3 claims it is 112.46, and source 4 claims it is 112.49. If a customer requires data accuracy down to the hundredth decimal place, four different value choices are available for honest voters to choose from since their values are similar. However, if customers can compromise on accuracy and settle for data accuracy down to the tenths place, the number of value choices that honest voters can select from is reduced to one, which is 112.5. With a lower accuracy requirement, data from various sources can have higher precision, and honest voters have fewer value choices to select from. As a result, most honest votes will be more concentrated around one choice, thus incrase the SPO's resistance against bribery attacks.

5.2 Cost Analysis

In our implementation, all computations are processed on-chain. Doing computations on-chain makes the result of the computation verifiable. However, it is also very costly since all computations on-chain cost fee in units of gas. In the SPO smart contract, the `FindResult()` method takes a lot of computing power to find a result for a voting game. We tested the gas cost of calling the `FindResult()` method for 6 different voting games. Each voting game had either a different vote count or a different standard deviation of votes compared to others. Table 3 show the vote count and the standard diviation for each voting game. The cost of calling the `FindResult()` method for each voting game is also shown in Table 3.

In Table 3 we can see that the cost of processing the `FindResult()` method is high. Game 4, which was the game that has the lowest cost to run the `FindResult()` method among all, would still cost 1588012 gasses to process the `FindResult()`

Table 3: Table of gas cost for running `FindResult()`

ID	Vote Count	Standard Deviation	Gas	Ether
1	500	2	2,071,403	0.107712956
2	500	5	5,307,977	0.276014804
3	500	10	9,842,971	0.511834492
4	100	2	1,588,012	0.082576624
5	100	5	3,373,440	0.17541888
6	100	10	6,901,718	0.358889336

method. The price of an ether on May 04, 2023, was roughly 1895.05 U.S. dollars [22]. Based on this price, the cost of processing the `FindResult()` method for game 4 would be 156.5 U.S. dollars.

From Table 3, we can also tell that the cost will increase whether when the standard deviation of votes has increased or when the vote count has increased. The gas cost increases proportionally as the standard deviation increases. However, the cost will increase slower as the vote count increases. Again, the `FindResult()` method is very costly to run. In the future, we will need a better solution to address the cost issue.

CHAPTER 6

CONCLUSION AND FUTURE IMPROVEMENTS

In this paper, we introduce a price oralce called Spartan Price oracle (SPO). Although the mechanism design of oracle largely follows the design of SchellingCoin, it has two significant improvements. The first improvement is to utilize the KDE method for determining the accurate data, which can prevent the game result from being manipulated by certian bribery attacks. However, if the percentage of compromised voters exceed a certain threshold, SPO will fail to prevent a bribery attack and can perform worse in comparison to SchellingCoin. Additionally, it utilizes a redistributive economic incentive model along with an appeal mechanism to significantly increase the cost of bribery. By implementing these two improvements, SPO becomes more resistent to bribery attacks.

SPO's major challenge is the high cost associated with running the voting game. Currently, all the computations are processed on-chain, leading to significant costs. It would be more efficient to run these computations off-chain in the future. One of the issues concerning off-chain computations is the selection of the participant responsible for performing these computations. Another concern revolves around verifying the results of the computations conducted by the participant on an off-chain computer. One possible solution to the first issue is to use a verifiable random function, as presented by Shuang et al. [23]. This function can randomly select a participant to perform the off-chain computation and post the result of the computation to on-chain storage. To verify the result, the SPO can initiate a dispute round that allows voters to vote on the accuracy of the result. If the result is considered false by the majority of voters, the SPO should impose a penalty on the participant for submitting the false result and select a new participant to perform the computation off-chain. By moving the computation to an off-chain computer, the cost of running a voting game in SPO

should be significantly reduced.

LIST OF REFERENCES

[1] S. Nakamoto, ''Bitcoin: A peer-to-peer electronic cash system,'' https://bitcoin.org/bitcoin.pdf, oct 2008, (Accessed on 10/01/2022).

[2] Z. Allam, ''On smart contracts and organisational performance: A review of smart contracts through the blockchain technology,'' *Review of economic and business studies*, vol. 11, no. 2, pp. 137--156, 2018.

[3] Z. He and L. W. Cong, ''Blockchain disruption and smart contracts,'' *The Review of Financial Studies*, vol. 32, no. 5, pp. 1754--1797, 2019.

[4] V. Buterin, ''Ethereum: A next-generation smart contract and decentralized application platform,'' https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf, 2014, (Accessed on 10/01/2022).

[5] H. Al-Breiki, M. H. U. Rehman, K. Salah, and D. Svetinovic, ''Trustworthy blockchain oracles: Review, comparison, and open research challenges,'' *IEEE access*, vol. 8, pp. 85 675--85 685, 2020.

[6] T. H. Pranto, A. A. Noman, A. Mahmud, and A. B. Haque, ''Blockchain and smart contract for iot enabled smart agriculture,'' *PeerJ. Computer science*, vol. 7, pp. e407--e407, 2021.

[7] A. Albizri and D. Appelbaum, ''Trust but verify: The oracle paradox of blockchain smart contracts,'' *The Journal of information systems*, vol. 35, no. 2, pp. 1--16, 2021.

[8] S. E. Chang, Y. Chen, and M. Lu, ''Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process,'' *Technological forecasting & social change*, vol. 144, pp. 1--11, 2019.

[9] T. Nugent, D. Upton, and M. Cimpoesu, ''Improving data transparency in clinical trials using blockchain smart contracts [version 1; peer review: 3 approved],'' *F1000 research*, vol. 5, pp. 2541--2541, 2016.

[10] J. krepelka, P. Collins, R. Cordell, S. Richards, I. E. Ashimine, P. Grimaud, and et al., ''Oracles,'' https://ethereum.org/en/developers/docs/oracles/, apr 2023, (Accessed on 05/01/2023).

[11] S. Alexander, ''Nash equilibria and schelling points,'' https://www.lesswrong.com/posts/yJfBzcDL9fBHJfZ6P/nash-equilibria-and-schelling-points, jun 2012, (Accessed on 10/01/2022).

[12] V. Buterin, ''SchellingCoin: A minimal-trust universal data feed,'' https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed, mar 2014, (Accessed on 10/01/2022).

[13] W. George and C. Lesaege, ''An analysis of p+ε attacks on various models of schelling game based systems,'' *Cryptoeconomic Systems*, vol. 1, no. 2, oct 22 2021, https://cryptoeconomicsystems.pubpub.org/pub/george-schelling-attacks.

[14] E. Parzen, ''On estimation of a probability density function and mode,'' *The Annals of Mathematical Statistics*, vol. 33, no. 3, pp. 1065--1076, 1962.

[15] V. Buterin, ''The p + epsilon attack,'' https://blog.ethereum.org/2015/01/28/p-epsilon-attack, jan 2015, (Accessed on 10/01/2022).

[16] ''Uma data verification mechanism: Adding economic guarantees to blockchain oracles,'' https://github.com/UMAprotocol/whitepaper/blob/master/UMA-DVM-oracle-whitepaper.pdf, apr 2020, (Accessed on 10/01/2022).

[17] J. Peterson, J. Krug, M. Zoltu, A. K. Williams, and S. Alexander, ''Augur: a decentralized oracle and prediction market platform (v2.0),'' https://whitepaper.io/document/29/augur-whitepaper, mar 2018, (Accessed on 10/01/2022).

[18] H. Huilgolkar, ''Razor network: A decentralized oracle platform,'' https://github.com/razor-network/whitepaper/blob/master/Razor%20Network%20Whitepaper%201.4.pdf, jan 2021, (Accessed on 10/01/2022).

[19] P. Sztorc, ''Truthcoin: Peer-to-peer oracle system and prediction market,'' https://www.truthcoin.info/papers/truthcoin-whitepaper.pdf, dec 2015, (Accessed on 10/01/2022).

[20] A. S. D. Pedro, D. Levi, and L. I. Cuende, ''Witnet: A decentralized oracle network protocol,'' https://arxiv.org/pdf/1711.09756.pdf, nov 2017, (Accessed on 10/01/2022).

[21] A. Gontijo, S. Richards, R. Cordell, E. Y. Yilmaz, H. Almidan, M. Scarset, and et al., ''ERC-20 token standard,'' https://ethereum.org/en/developers/docs/standards/tokens/erc-20/, apr 2023, (Accessed on 05/01/2023).

[22] https://ethereumprice.org/, may 2023, (Accessed on 05/04/2023).

[23] S. Yao and D. Zhang, ''An anonymous verifiable random function with applications in blockchain,'' *Wireless Communications and Mobile Computing*, vol. 2022, pp. 1--12, 2022.