Improvements to Incentives in Incentive-Driven Blockchain Technologies

Conor McMenamin

January, 2023

Universitat Pompeu Fabra *Barcelona*

TESI DOCTORAL UPF / 2023
Supervisors: Vanesa Daza and Matteo Pontecorvi
Department of Information and Communication Technologies

Thanks

Throughout my PhD, many people have come and gone, but the important ones willed out. First, to my family and friends who have been with me throughout the PhD. They say a journey shared is a journey halved, but this journey defies the norm. Little did you know that every conversation, every coffee, every hug, every night out, every pádel match, every evening in the Palau Blaugrana, every Copa Catulunya, every Valheim resource grind, every board game, everything; it was all carrying me to this point. In the microcosms of struggle that typically mar a PhD, you got me back to my best. The past three years and change have been some of the most enjoyable of my life. Time has flown, and the journey is better described as a dream. This is in large thanks to you all. Go raibh míle míle maith agaibh.

Thanks to Matteo, Matthias, Padraic, Bruno, and my extended network of collaborators. Your insights and discussions have played a significant role in the PhD, and hopefully this is only the beginning of what has already been a rewarding and fruitful collaboration. Many thanks must go to Samuel for tailoring such a haute couture PhD experience while at Nokia. The fit could not have been better.

The final, and possibly most important thanks of all must go to Vanesa. You have been at the core of everything good in my PhD. Much of my thesis is based around the hypothesis that people typically seek to maximize their own utility. This makes you an exception to the rule, repeatedly and unselfishly seeking to ensure I had everything I needed to get through the PhD. This is something for which I will be eternally grateful, and an approach I hope to emulate as I continue along my path in life. In this sense, I hope imitation indeed serves as the highest form of flattery Vanesa, as words may never be enough. Thank you.

Funding

This thesis is part of a project that has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement number 814284

Abstract

In this thesis, we present a set of protocols in areas at the core of current blockchain technology literature; consensus, decentralized finance and distributed computing. These areas are bound by a critical dependency on incentivization. Despite this, existing protocol standards in each of these areas are vulnerable to well-documented incentivization exploits which limit their attractiveness compared to centralized alternatives. We first identify the shortcomings of existing standards in these areas with respect to incentivization. We then take a common, general approach to these shortcomings, and in each instance, propose a novel protocol with which to address the incentivization shortcomings we identify.

Resum

Aquesta tesis presenta una sèrie de protocols en àrees que pertanyen al nucli de la tecnologia Blockchain com són els algorismes de consens, les finances descentralitzades o la computació distribuïda. Aquestes àrees estan unides per una dependència crítica dels incentius. Malgrat això, els estàndards dels protocols existents en cadascuna d'aquestes àrees són vulnerables en l'ús que fan dels incentius, fet que limita el seu atractiu en comparació amb les alternatives centralitzades. En aquesta tesi identifiquem primer els defectes dels estàndards existents en aquestes àrees pel que fa als incentius. A continuació, aproximem de manera comuna i genèrica aquests defectes i per a cada escenari, introduim un protocol que evita les mancances detectades als protocols existents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our deepest fear is not that we are inadequate, our deepest fear is that we are powerful beyond measure.

Marianne Williamson

Incentivization can be seen as a symbol of the duality that exists within the blockchain technology space. On one hand, it is the innovation the sparked an explosion in blockchain technology. The novel application of incentives in the now seminal Bitcoin whitepaper [85] beckoned a golden age for distributed-systems research driven by the popularity of cryptocurrency. In the Bitcoin whitepaper, Satoshi Nakamoto is immortalized through his proposal to reward players in expected proportion to provable work done in maintaining and updating the underlying blockchain. This provided the missing piece to decades of distributed systems literature seeking to establish a truly permissionless transaction ledger.

On the other hand, incentivization can be seen as a topic in the blockchain technology space where innovation is most needed. Following the identification of several shortcomings in Nakamoto's model as early as [48, 52], the space was primed for new advancements building on Nakamoto's key contribution of proof-of-work incentivization. Unfortunately, incentivization research in the years since has largely regressed to pre-Nakamoto assumptions of honest-by-default players who ignore incentives and always follow prescribed protocols.

More than this, applications using these shaky consensus-level foundations as a trusted transaction ledger have been costing their users hundreds of millions of dollars due to the misalignment of incentives. In the decentralized world of blockchain technology, resource advantages such as better connectivity to other players, or more hash-power, can allow certain players to monopolize the right to order and even include or exclude transactions, powers whose abuse blockchain applications incentivize. Although these phenomena are being observed and documented [93, 43], provably secure solutions are lacking [58]. This is the primary motivator for the thesis.

At the consensus level, this disconnect is highlighted in Fruitchains [92], one of the most referenced academic works in relation to fairness of reward distribution and incentive compatibility. Fruitchains crucially relies on an underlying blockchain satisfying a state machine replication (SMR) protocol in order to guarantee fairness of rewards on the chain distributing the rewards. The authors fail to consider the incentives of all parts of the system, relying on an altruistic majority of players participating in an underlying consensus protocol to fairly reward all players for the work they have done. The fallacy in this reasoning can be seen through the simple strategy of only rewarding oneself and not rewarding other players in the Fruitchains protocol (not pointing to other players' "fruits"). This strategy strictly dominates the Fruitchains strategy with respect to maximizing rewards share. Oversights like this are ingrained at the foundations of distributed-system incentivization research, and it is these oversights which stand as the focus of this thesis.

At the core of the issues we identified in consensus-level incentives was the dependency on one or more altruistic, honest-by-default players (see Section 3.2 for an extended summary of these findings). Within the foundational BAR (Byzantine, Altruistic and Rational) player model [6] to which distributed systems literature on incentives typically reference, such a dependency is allowed. This motivates the first contribution of the thesis. In Chapter 3, we propose the *ByRa player model*, a player model free from altruistic player dependencies within which incentives are required to be more robust. Within this new player model, we propose Tenderstake, a consensus-level protocol which is incentive compatible under a rational majority and adversarial minority of players, and importantly, without any need for honest players.

Beyond consensus-level incentives, application-level incentives in blockchain-based systems are not themselves without flaw, and form the basis for the remainder of the thesis. The most prominent research on blockchain-based application-level incentives can almost surely be traced back to the now seminal work of Flashboys 2.0 [43]. In Flashboys 2.0, the authors examine the smart-contract enabled Ethereum blockchain at a transaction level. The authors identify that the block-producer privilege of being able to order, censor and create transactions is highly profitable, beyond the base block rewards for creating valid blocks. This profitability was coming at the expense of the users submitting transactions for inclusion in the blockchain, in the now infamous phenomenon of *MEV* (then miner extractable value, now maximal extractable value as this value extraction is not necessarily exclusive to the miner).

Although the incentives in mainstream blockchains were misaligned at the consensus-layer (as demonstrated in Selfish Mining [48]), the price of anarchy [83] for deviating from the prescribed protocol and reducing trust in the underlying blockchain seemed to be ensuring that consensus-level incentives were performing more-or-less as required. However, as evidenced in [93, 43], it was clear that application-level incentives were not. The foundational example for this can be seen in the Uniswap decentralized exchange (DEX) protocol [102]. An automated market-maker, accessible by all, allowing swaps from one token to almost any other was a simple, seemingly effective public good, yet overall limited, and deceptively expensive to interact with. Announcing an intent to trade in an exact size, price and direction before it gets confirmed, as is done in Uniswap and most other DEX implementations, is highly exploitable in a competitive trading environment. The quantified losses sustained by users submitting transactions to Uniswap are in the hundreds of millions of dollars [93, 50]. Exact figures are impossible to quantify, but are bounded below by these recorded losses. The total value extracted from users across all protocols and blockchains is likely therefore to be in the billions of dollars.

This can be seen as a much more pressing issue than consensus-level incentives with no glaring ongoing exploits, and one in which research is identified as lacking [58, 72]. At the core of MEV is the informational advantage held by some subset of players in the system when deciding what action to take. If a player can predict with some probability greater than random how the underlying state machine will update (by observing the transactions likely to be accepted in the coming blocks for example), players' strategies change dramatically. In the DEX example, a player aware of an imminent buy imbalance for a particular token in upcoming blocks gives that player the ability to front-run the information, and/or back-run the information when this buy imbalance is likely to cease. This issue forms the basis for Chapters 5 and 6. In these chapters we highlight this advantage, and construct protocols under the assumption that this information will be used. In this paradigm, we propose two new DEX variants which protect uninformed users from the negative externalities of such informational disadvantages.

With respect to randomness-based selection, and/or rewarding based on popular choice, key components in distributed computation (DC) outsourcing, the problem is no different. Unfortunately, research in the area is significantly more detached from the realities of incentives, focusing on scalability, speed, and applications, despite computation outsourcing being a billion dollar privatized industry [8]. For decentralized DC to compete with such a behemoth, incentivization is needed. There are a litany of works related to DC incentives [71, 18, 64, 103] which fail to capture a world in which computers must be rewarded financially for computations, instead depending on computer utility to be measured in something directly related to the quality of the computations they are performing. Like using an SMR protocol to incentivize SMR as in [92], this is another incentives chicken-and-egg scenario, rendering the potential of DC moot in the real world. In Chapter 4 we address some of the major gaps in these previous works on incentivization in DC, providing a generic protocol for blockchain-based DC which is incentive compatible in the ByRa model.

Following on from the foundation on incentives and blockchain formalization provided in Chapter 3, the remainder of the thesis can be seen as a road-map to tackle some of the key open problems in decentralized protocol incentives. This is the backdrop for Chapters 4, 5 and 6 in which we provide incentive-specific improvements on industry standards with regards to distributed computing, decentralized exchange and automated market-makers respectively. The contributions of all chapters can be summarized as follows.

- Chapter 3: Tenderstake. We outline the ByRa model, a new player model free of altruistic players, and argue that this player model more accurately describes players in distributed games. We outline the properties of strong incentive compatibility in expectation and fairness as properties that distributed consensus protocols must possess in the ByRa model in order to guarantee state machine replication. We then provide Tenderstake as a protocol which possesses these properties.

- Chapter 4: Marvel DC. We present Marvel DC, a blockchain-based strong incentive compatible distributed computing protocol. Marvel DC stands as an improvement on existing industry standards in which computers are altruistic, or trust third parties (TTPs) are required to ensure honest behaviour. We also provide Privacy Marvel DC, a privacy enhancement for Marvel DC which decouples computation outputs from the computers who computed them, making Privacy Marvel DC appropriate for computations in which outputs potentially reveal sensitive information about the computers computing them, such as Federated Learning.

- Chapter 5: FairTraDEX. We present FairTraDEX, a decentralized exchange protocol based on frequent batch auctions in which the game-theoretic optimal strategy for all players is to trade at the true market-implied price of the underlying token swap, excluding explicit fees.

- Chapter 6: Diamond. We present Diamond, an automated market making protocol that aligns the incentives of liquidity providers and block producers in the protocol-level retention of losses-versus-rebalancing [82] (loss-versus-rebalancing is explained in Section 6.3.2). In Diamond, block producers effectively auction the right to capture any arbitrage that exists between the external market price of a Diamond pool, and the price of the pool itself. The proceeds of these auctions are shared by the Diamond pool and block producer in a way that is proven to remain incentive compatible for the block producer.

Organization. The thesis starts with a chapter on preliminaries and background knowledge that is used throughout the proceeding chapters, but contains no new results. Afterwards, each chapter corresponds to one paper, some published and some in pre-print stage under review for publication. The contents are essentially the same as in the papers, with only minor modifications to remove redundancies, unify notation and ensure a more cohesive document.

Chapter 2

Preliminaries

This chapter introduces the terminology and definitions necessary to understand the main results of the thesis. By *negl*() we denote any function $f : \mathbb{N} \rightarrow \mathbb{R}$ that decreases faster than any (positive) polynomial p. More formally, $\forall\ p\ \exists\ \lambda_0 \in \mathbb{N}$: $\forall \lambda > \lambda_0 : f(\lambda) < \frac{1}{p(\lambda)}$. In all chapters, for protocol correctness we assume that some of the involved players may be malicious trying to force the protocol into incorrect execution, and without any direct benefit for themselves. However, for the game-theoretic part of the analysis, we assume that a majority of players are rational, at all times trying to maximize some utility function. Accordingly, the analysis in our thesis is based on two security parameters, a cryptographic security parameter κ used to bound the probability that certain protocol executions are incorrect, and a game-theoretic security parameter ψ. For now, it is sufficient to consider the game theoretic parameter as bounding the probability that one player has an advantage over another with regard to some common utility function. When ψ is used, the game-theoretic probability it is bounding is specified.

All references within a chapter to lines or algorithms in provided protocol encodings are specific to the encodings within that section.

2.1 State Machine Replication & Blockchain

In this thesis, we are typically interested in a distributed set of players $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ interacting with one and other inside a protocol which will produce some output that all players correctly participating in the protocol can agree on. This output will be a *replicated state machine.* First, we define a state machine.

Definition 2.1.1. *A state machine consists of set of variables, and sequence of commands/ updates on those variables, producing some output.*

The concept of a state machine alone does not capture the notion that potentially many players can reconstruct a common view of the same state of a machine, and requires extension.

Definition 2.1.2. *For a set of players $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ and a state machine, state machine replication (SMR) is a process that allows each player to execute a common sequence of commands acting on the machine's state in the same order, thus maintaining a common view of the machine's state.*

Progressing towards our goal of analysing SMR protocols, we must first define what we require from an SMR protocol. We take inspiration for our definition from [1], where their system model is clearly and concisely explained, and is very similar to ours.

Notation 2.1.3. *With respect to protocols and recommended protocol actions, a correct player is a player who always follows the recommended protocol actions.*

Definition 2.1.4. *An SMR protocol Π deciding on a potentially infinite sequence of state machine updates satisfies the following properties:*

- *Safety: For any two correct players $\mathcal{P}_i$, $\mathcal{P}_j$ in Π, $i \neq j$, if $\mathcal{P}_i$ decides on an SMR update V_i at position k in the sequence, and $\mathcal{P}_j$ decides on an SMR update V_j at position k in the sequence, then $V_i = V_j$.*
- *Liveness: For any position k in the sequence, every correct player eventually decides on an SMR update for position k.*

To achieve SMR, we utilise the concept of a blockchain. This is done in a generic manner so as to allow for direct comparison with most blockchain instantiations.

Definition 2.1.5. *A block B is a data structure used to communicate changes to the state machine view of each player. Blocks consist of a pointer(s) to previous block(s), and a set of instructions with which to update the state. State machine updates in a block are applied to the state described by the block(s) to which they point. The genesis block B^1 describes the starting state of the system and is a priori agreed upon by all players. The global state at any point in the system is then described by applying the state machine updates according to* *some ordering rule starting from the genesis block. A blockchain Blockchain* $= [B^1, ..., B^{\mathcal{H}}]$ *is the ordered data structure created by traversing the block pointers from the genesis block to all blocks to be applied to the global state according to the ordering rule.* $\mathcal{H}$ *denotes the height of the blockchain.*

2.2 Game-Theory

This section introduces some basic game theory to allow us to properly reason about SMR protocols in our system as games, taking inspiration for our definitions from [88]. The games we are concerned with, SMR protocols, are played by players with strict incomplete information, meaning some subset of players will not know the action choices of other players for certain rounds when they are required to choose their own actions. As such, we need to be able to describe what a player knows (and implicitly what they do not), which we call their private information. Furthermore, we must be able to describe what motivates players in games. This motivation is provided by a utility function, which attributes a numerical score to each action a player can take. In games, players choose the action which maximises their utility function.

Definition 2.2.1. *A game, denoted* $\mathbb{G}$*, progressing in rounds with strict incomplete information for a set of n players* $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ *can be described by the following:*

- *For every* $\mathcal{P}_i$*, a set of actions* $\mathbb{X}_i$*. We denote by* $\mathbb{X}_{-i}$ *the set of actions that each player excluding* $\mathcal{P}_i$ *can take. For* $\mathcal{X}_{-i} \in \mathbb{X}_{-i}$*,* $\mathcal{X}_{-i}$ *is described by a vector of actions of length* $n-1$*, with each vector position mapping to a unique player.*

- *For every player* $\mathcal{P}_i$ *and round* r*, a set of private informations* T_i^r*. A value* $t_i^r \in T_i^r$ *is a private information value that* $\mathcal{P}_i$ *can have at round* r*. We denote by* t_{-i}^r *the private informations held by all players excluding* $\mathcal{P}_i$ *at round* r*.*

- *For every player* $\mathcal{P}_i$*, current round* $r \geq 1$*, and some round* $r' \geq r$*, the utility function for* $\mathcal{P}_i$ *with respect to round* r' *is defined as :*

$$u_i^r : T_i^r \times \underbrace{\mathbb{X}_i \times ... \times \mathbb{X}_i}_{r'+1-r} \times \underbrace{\mathbb{X}_{-i} \times ... \times \mathbb{X}_{-i}}_{r'+1-r} \rightarrow \mathbb{R} \tag{2.1}$$

where $u_i^r(t_i^r, \mathcal{X}_i^r, ..., \mathcal{X}_i^{r'}, \mathcal{X}_{-i}^r, ..., \mathcal{X}_{-i}^{r'})$ *is the utility achieved by* $\mathcal{P}_i$ *in round* r' *with private information* t_i^r*, if player* $\mathcal{P}_i$ *takes the actions* $\mathcal{X}_i^r, ..., \mathcal{X}_i^{r'}$

in rounds $r, ..., r'$ respectively, and the actions of all other players are described by $\mathcal{X}_{-i}^{r}, ..., \mathcal{X}_{-i}^{r'}$ in rounds $r, ..., r'$ respectively.

Although utility functions evaluate actions given the actions of all other players, the actions of the other players may not be known in advance. Therefore, players will need to be able to choose their actions solely based on their private informations. The actions a player takes given some private information are computed through a strategy, which is defined in Definition 2.2.2.

Definition 2.2.2. *A strategy of a player $\mathcal{P}_i$ is a function $str_i : T_i^r \rightarrow \mathbb{X}_i$, $r \geq 1$, which defines the action to be taken by $\mathcal{P}_i$ given some private information value. A strategy str_i is mixed if for a player $\mathcal{P}_i$ with m_i possible strategies $Str_i = \{str_i^1, ..., str_i^{m_i}\}$, they select a strategy to follow from Str_i according to some probability distribution. For every player $\mathcal{P}_i$, str_{-i} describes the mixed strategies taken by all players excluding $\mathcal{P}_i$.*

In this thesis, we also utilize the well-studied concept of strict Nash equilibria, defined as follows.

Definition 2.2.3. *Consider a set of non-cooperative players $\mathcal{P}_1$,..., $\mathcal{P}_n$, with strategies str_1 ,..., str_n describing the actions which each player takes throughout a particular protocol. These strategies form a Strict Nash Equilibrium if any individual player deviation from these strategies strictly reduces that player's utility.*

2.3 Non-Interactive Zero-Knowledge Set Membership

The aim of this section is to outline existing *non-interactive zero-knowledge* (NIZK) tools for proving set membership, such as those stemming from papers like [81, 23, 57, 25, 59]. We define these tools generically, allowing for the adoption of any secure NIZK set-membership protocol, as we only require a common functionality that is shared by all of them.

Proving membership has been traditionally solved using cryptographic accumulators [24], where a *prover* computes a value (the accumulator) and, based on this value, a set of short membership proofs that a *verifier* can easily verify. Three of the main approaches to construct set membership proofs are Merkle trees [80], RSA accumulators [19, 30], and pairing-based accumulators [87, 105].

Each approach has its own benefits for public parameters, accumulator or witness size or need of trusted setup. The exact choice depends on the resource constraints of the system. We direct interested readers to [25] for an extended review of the main features of each of these approaches.

When the prover does not want to reveal a value x, the membership proof of x in the accumulator should not leak any information on the value of x. At a high level, the general approach is to guarantee privacy using zero knowledge proofs. Zero knowledge proofs [56] are powerful cryptographic primitives that allow a prover P to prove knowledge of the truth of some statement without revealing the statement contents, to some honest verifier V who needs to be convinced of the truth of the statement provided by the prover. In this thesis, we are specifically interested in zero-knowledge proofs that are also non-interactive, that is, proofs that only depend on the prover's private information about the statement and publicly available information. This public information can come in many forms, but in [81, 23, 57, 25], it must be generated honestly in a process known as a *trusted setup.* If a prover knows the private information used to generate public proof parameters, the knowledge extraction property cannot exist.

As such, NIZK proofs do not depend on interaction with the verifier. The main features in a NIZK argument are completeness, soundness, and zero-knowledge. Completeness guarantees that if the statement is true, the prover behaving honestly can convince the verifier that the statement is true, while soundness ensures that a dishonest prover cannot convince an honest verifier. Zero-knowledge maintains that the only information learned by the verifier is that the statement is true. However, in practice it is required that the prover knows a witness for the statement, that is, a zero-knowledge proof of knowledge. In this case, soundness is not enough and is required that a prover cannot produce a valid proof unless she knows a witness, even if the prover has seen an arbitrary number of simulated proofs. Furthermore, NIZK arguments are interesting for constructing other cryptographic primitives, such as a signature of knowledge (SoK) [37].

The NIZK proving of set membership typically involves some combination of *Merkle Trees* and *RSA accumulators*, as well as *NIZK succinct arguments of knowledge* (*NIZK-SNARKs*). The use of RSA accumulators such as that described in [81] has been rather thoroughly replaced by NIZK-proofs on Merkle-tree accumulators [23, 59]. More recently, hybrid approaches have emerged [25]. The exact choice of NIZK tools depends on the resource constraints of the system. RSA accumulators favour dynamic set sizes, less public parameters and reduced prover computation time in general, while Merkle Trees and NIZK-SNARKs generally favour short verification time and proof size.

Informally, the NIZK proofs used in this thesis allow, for a given set of commitments *Com* to user-generated secrets, that any user knowing the secret corresponding to a commitment $com \in Com$ can prove the knowledge of a secret corresponding to a commitment in the set, without revealing which secret, or commitment. Moreover, we require that more than one proof relating to the same commitment is identifiable by a verifier.

We first provide a high-level outline of the key properties that we require from a NIZK proof-system in line with the existing industry standards utilised and proved to exist in the above protocols:

- **Completeness**: If the statement is true, the prover acting honestly can convince the verifier that the statement is true.
- **Knowledge Extraction**: If the statement is false, the prover cannot convince the verifier that the statement is true.
- **Zero-Knowledge**: If the statement is true, the only information learned by the verifier is that the statement is true.
- **Non-Interactive**: Proofs only depend on the prover's private information about the statement and publicly available information[1]. As such, proofs do not depend on interaction with the verifier.

We now describe the tools more precisely. We define a commitment scheme *commit*, a set-membership proof scheme *SetMembership*, an NIZK proof of knowledge scheme *NIZKPoK* and a NIZK signature of knowledge scheme (*NIZK-SoK*). We do not specify which instantiation of these schemes to use, as the exact choice will depend on several factors, such as efficiency, resource limitations and/or the strength of the assumptions used. Provers in this thesis privately generate two bit strings, the *serial number* $\mathcal{S}$ and *randomness* $\mathcal{R}$, with $\mathcal{S},\ \mathcal{R} \in \{0,1\}^{\Theta(\kappa)}$. Provers then commit to these values by generating a commitment $com \leftarrow commit(\mathcal{S},\mathcal{R})$. This *com* is then broadcast to the verifier, with the set of all commitments denoted by *Com*. It is this set of commitments *Com* to which the prover seeks to prove membership.

- $commit(m)$: A deterministic, collision-resistant function taking as input a string $m \in \{0,1\}^*$, and outputting a string $com \in \{0,1\}^{\Theta(\kappa)}$.
- $SetMembership(com, Com)$: Compresses a set of commitments *Com* and generates a membership proof π that *com* is in *Com* if $com \in Com$.

[1]This public information can come in many forms, but in [81, 23, 57, 25], it must be generated honestly in a process known as a *trusted setup*. If a prover knows the private information used to generate public proof parameters, the knowledge extraction property cannot exist.

- $NIZKPoK(\mathcal{R}, \mathcal{S}, Com)$: For a set of commitments Com, returns a string $\mathcal{S}$ and NIZK proof of knowledge that the player running $NIZKPoK()$ knows an $\mathcal{R}$ producing a proof when running $SetMembership(commit(\mathcal{S}||\mathcal{R}), Com)$. This revelation identifies to a verifier when a proof has previously been provided for a particular, albeit unknown, commitment as the prover must reproduce $\mathcal{S}$. This is used in the thesis to enforce the correct participation of players.

- $NIZKSoK(m)$: Returns a signature of knowledge that the player who chose m can also produce NIZKPoK.

2.4 Relayers

A fundamental requirement for transaction submission in blockchains is the payment of some transaction fee to simultaneously incentivise block producers to include the transaction, and to prevent denial-of-service/spamming attacks. However, in both the unspent transaction- and account-based models, this allows for the linking of player transactions, balances, and their associated transaction patterns. With respect to DEX protocols, if clients are required to deposit money into a unspent transaction/account before initiating a trade, any other player in the system can infer who the client is, what balances the client owns, what transactions the client usually performs, etc., and use this information to give the client a worse price.

To counteract this, we utilise the concept of *transaction relayers*, such as those used in 0x[2], Open Gas Station Network[3], Rockside[4], and Biconomy[5]. In the FairTraDEX and Privacy Marvel DC protocols, users must publicly register to a smart contract, and in doing so, deposit some escrow. In addition to this escrow, we also require the users to deposit a relayer fee. When the user wishes to submit a transaction anonymously to the blockchain, the user publishes a proof of membership in the set of registered users to the relayer mempool, as well as the desired transaction and a signature of knowledge cryptographically binding the membership proof to the transaction, preventing tampering. As the relayer can verify the proof of membership, the relayer can also be sure that if the transaction is sent to the protocol contract, the relayer will receive the corresponding fee. With this in mind, a relayer observing the user transaction

[2]0x https://0x.org/docs/guides/v3-specification

[3]Open Gas Station Network https://docs.opengsn.org/

[4]Rockside https://rockside.io/

[5]Biconomy https://www.biconomy.io/

includes it in a normal blockchain transaction, with the first relayer to include the transaction receiving the fee. As such, relayers are a straightforward extension of the standard transaction-submission model. Furthermore, if the proof of membership is NIZK and the message is broadcast anonymously (using the onion routing (Tor) protocol[6] for example), the relayer can only infer that the player sending the transaction is a member of the set of users.

2.5 Financial Terminology

This section contains an overview of financial terms and definitions as they are used in Chapters 5 and 6.

- *Limit Order*: Specifies an amount of tokens to be bought (sold), and a maximum (minimum) price at which to buy (sell) these tokens. This price is known as the *limit price*.

- *Market Order*: Specifies an amount of tokens to be sold, but no limit price. Market orders are to be executed immediately at the best available price based on the liquidity of buy orders.

- *Direction*: With respect to an order on a market quoted from token x to y, if the order is trying to buy token y, the direction is *buying*, while if the order is trying to sell token y, the direction is *selling*.

- *Forward Price*: This is the price at which a seller delivers a token to the buyer at some predetermined date. In any exchange protocol without instantaneous delivery, the forward price at expected delivery time is the price at which trades should happen. The difference between current (spot) price and forward price is known as *carry*, and can be due to storage/opportunity costs, interest rates, etc. In this thesis, we set carry to 0.

- *Notional Value*: The value of a set of tokens expressed in some common reference token. In this thesis, we use the symbol ₿ as the reference token in which we measure notional, and with which we reason about utility.

- *External Market Price* (denoted ε): As in [33], the external market price of a token/token swap is a publicly observable signal which is perfectly informative of the fundamental price of the underlying token/token swap.

[6]https://www.torproject.org/

Moreover, a random order of fixed notional generated by a player in the system is equally likely to buy or sell tokens at the external market price, distributed symmetrically around the external market price. Unless otherwise stated, observing the external market price has a prohibitive cost for players in our system.

- *Market*: A market in a DEX between two tokens x and y consists of two limit orders, a *bid* and *offer*. When the market is quoted from token x to y, the offer price indicates the quantity of token x a player must sell for 1 token y, while the bid price indicates the quantity of token x a player receives for 1 token y. In this thesis, we represent such a market as *bid* @ *offer*, with $0 < \mathit{bid} \leq \mathit{offer}$.
- *Reference Price* (ε_{ref}): For a market *bid* @ *offer*, the reference price ε_{ref} is the price such that $\frac{\mathit{bid}}{\varepsilon_{ref}} = \frac{\varepsilon_{ref}}{\mathit{offer}}$, i.e., ε_{ref} is the geometric mean of *bid* and *offer*.
- *(Market) Width* (*width*): For a market *bid* @ *offer*, the width is calculated as $\mathit{width} = \frac{\mathit{offer}}{\mathit{bid}}$ (as such $\mathit{width} \geq 1$).
- *Multiplicative Market-Impact Coefficient (δ)*: If the pre-trade external market price for particular swap is ε, the expected post-trade external market price given a buy order is $\delta\ \varepsilon$ for some $\delta \geq 1$, while the expected post-trade external market price given a sell order is $\frac{\varepsilon}{\delta}$. Unless otherwise stated, a swap from x to y with multiplicative market-impact coefficient δ corresponds to buy orders of y having a multiplicative market-impact coefficient on ε_y of $\sqrt{\delta}$ and $\frac{1}{\sqrt{\delta}}$ on ε_x. Given our definition of the external market price, this impact function implies an upward drift in ε if $\delta > 1$. However, our use of δ is intended to highlight that impact must be considered, with the exact choice of δ for a particular token pair being a complex process and beyond the scope of this thesis.
- *User*: Any player in a DEX protocol who, for an external market price ε, there exists some *minimum user utility fee* > 1 such that user buyers (sellers) have positive expected utility to trade for or below $\sqrt{\mathit{fee}}\ \varepsilon$ (at or above $\frac{\varepsilon}{\sqrt{\mathit{fee}}}$).
- *Market Maker* (MM): A player in a DEX protocol with large supplies of all tokens, who has positive expected utility trading with users on markets of any width $\mathit{width} > 1$ with reference price equal to the external market price. MMs can observe the external market price.

- *Automated Market Maker* (AMM): A DEX protocol with reserves of 2 or more tokens allowing any player with access to the AMM to swap tokens directly with the reserves of the AMM without the interaction of another player. As such, AMMs typically update their price after each interaction, adjusting to the previous swap. The most common method for updating price is keeping some pre-defined function of the reserves constant, known as a *constant function market maker* (CFMM). CFMMs are described in detail in Section 6.3.1.

Chapter 3

Tenderstake

This chapter is based on the paper *Achieving state machine replication without honest players* [79].

3.1 Introduction

Existing standards for player characterization in tokenized SMR protocols depend on honest players who will always follow the protocol, regardless of possible token increases for deviating. Given the ever-increasing market capitalization of these tokenized protocols, honesty is becoming more expensive and more unrealistic. As such, this out-dated player characterization must be removed to provide true guarantees of safety and liveness in a major stride towards universal trust in SMR protocols and a new scale of adoption. As many SMR protocols are built on these legacy standards, it is imperative that a new player model is identified and utilized to reflect the true nature of players in tokenized protocols, now and into the future.

In Flash Boys 2.0 [43] and the follow-up Flashbots project [50], it is demonstrated that protocol-deviation opportunities are rampant in Ethereum, and that players are actively availing of them. In any large-scale SMR protocol, most, if not all players, will not consider their deviations as affecting SMR. Therefore, it is essential that we assume non-adversarial players will seek to maximize tokens in tokenized protocols. As a direct consequence, SMR guarantees can no longer depend on honest-by-default users. We explicitly outline the ByRa (Byzantine or Rational) model as an updated player characterization framework to reflect this weakness in current standards. By moving to the ByRa model, which we formally define in Definition 3.4.1, the caveat of honest player dependencies in current SMR protocols is removed. Furthermore, we demonstrate that it is possible to achieve SMR in the ByRa model by providing the Tenderstake protocol, an amendment to the Tendermint protocol [68, 31].

To progress towards global adoption, a tokenized SMR protocol must first ensure that all players will maximize their tokens by following the protocol. Implementing an SMR protocol that increases a player's tokens for following the protocol is known as incentivization, and is a fundamental requirement for any SMR protocol. Much of the work on incentivization in SMR protocols stems from the seminal work on *selfish mining* in Nakamoto-consensus [48]. In [48], it is demonstrated that certain players are incentivized to deviate from the prescribed protocol. This eventually leads to a scenario where SMR properties are violated, as discussed in [48]. It is only upon the performing of actions as required by the protocol by some majority that it is possible to guarantee the SMR properties of safety and liveness. This has remained the case in the age of tokenization.

Despite this, there has been no thorough treatment and analysis of tokenized SMR protocols from a game-theoretic standpoint involving rational players, who want to maximize a known utility function, and an adversary, who can corrupt the owners of some amount of the tokenized consensus resource and behave arbitrarily. These corrupted players are known as Byzantine. This characterization of players as either Byzantine or Rational, which we refer to as the ByRa model, was first considered in distributed systems literature in [83], with recent advocates including [69, 9, 99]. The closest semblance to this model which has seen wide-scale adoption with respect to SMRs is the BAR model [6]. The BAR model crucially includes some portion of altruistic players who disregard tokenized utility, and always follow the protocol. Examples of authors echoing our desire to move away from altruistic dependencies are numerous, but this from Fairledger [69] puts it concisely: "We have to take into account that every entity may behave rationally, and deviate from the protocol if doing so increases its benefit". Non-adversarial, honest-by-default characters do not exist in competitive games, and cannot be depended on in tokenized SMR protocols due to their gamified nature. Although many other works state the need to move away from altruistic dependencies, none have proven the critical nature of this dependency, or provided protocols which achieve SMR, in the ByRa model. In this chapter, we fulfil both of these essential tasks.

Without the safety net of altruistic players, any successful instantiation of an SMR protocol in the ByRa model must guarantee that rational players will always follow the protocol. To ensure this, rational players must expect to strictly maximize their utility by following the protocol, a property we define as *strong incentive compatible in expectation* (SINCE).

Moreover, we must also guarantee that within such an incentive compatible protocol, the adversary cannot increase their share of tokens to a point where they control enough tokens to prevent SMR. Despite the existence of strong incentive compatibility in expectation, it may be possible for an adversary to receive more than their share of the tokens that get distributed, increasing their share of control. Therefore, we must additionally ensure that an adversary cannot increase the share of tokens they control, a property we define as *fairness.*

3.1.1 Our Contribution

We define the ByRa player characterization model, the properties of SINCE and fairness, and in Definition 3.4.4, the basic requirements a prospective SMR protocol must meet in order to guarantee safety and liveness in the ByRa model. If these requirements are met for a protocol in the ByRa model, the protocol *achieves ByRa SMR.* Informally, to achieve ByRa SMR we require that rational players control a majority of tokens at all times, and are always incentivized to follow the protocol. We then prove that the properties of SINCE and fairness are necessary and together sufficient to achieve ByRa SMR in the main theorem of the chapter.

Theorem 3.5.8. *For an SMR protocol* Π*,* Π *achieves ByRa SMR if and only if* Π *is strong incentive compatible in expectation and fair.*

In addition to this new game-theoretical framework, we provide Tenderstake as a concrete instantiation of an SMR protocol that provably achieves SINCE and fairness in the ByRa model. Using Theorem 3.5.8, we then prove Tenderstake achieves SMR in the ByRa model.

3.1.2 Organization of the chapter

In Section 3.2 we review related work and present an overview of attempts to implement, and works in favour of, the ByRa model for SMR protocols. In Section 3.3 we provide the background needed to define the ByRa model. Section 3.4 introduces a new game-theoretic framework for analysing SMR protocols. This new framework defines the ByRa model, and outlines what we require from SMR protocols in the ByRa model, introducing the properties of SINCE and fairness. In Section 3.5 we prove that SINCE and fairness are necessary

Protocol	Network Model	Player Model w/o Honest Players	Evolving-Stake Adversary	SINCE	Fair
Rationals vs. Byzantines [9]	Broadcast Synchrony[1]	✓	✗	✓ [2]	✓ [3]
Blockchain Without Waste [98]	Synchrony	✓	✗[4]	✓ [5]	✗
Blockchains Cannot Rely on Honesty [99]	Synchrony	✓	✓	✗	✗
Fruitchains, Snow White [92, 42]	Partial Synchrony	✗	✗	✗	✗
Casper Incentives [34]	Partial Synchrony	✗	✗	✗	✗
FairLedger [69]	Synchrony	✗	✗	✗	✗
Tenderstake	Partial Synchrony	✓	✓	✓	✓

Table 3.1: Comparison of consensus protocols claiming incentive compatibility (prior to publication of [79]). [1]An idealized model where every message, including adversarial messages, are known to be instantly delivered to all players. [2]No explicit reward mechanism provided, non-trivial for BFT protocols. [3]Enforced by the idealized network model/ unspecified reward mechanism. [4]No adversary in player model. [5]Author creates a dominating cost unrelated to quantity of stake for deviation.

for a protocol to achieve ByRa SMR. We then prove that together, SINCE and fairness are sufficient properties for a protocol to achieve ByRa SMR. In Section 3.6 we outline the Tenderstake protocol as an example, for the first time in literature, of a SINCE and fair ByRa SMR protocol. In Section 3.7 we reason that Tenderstake satisfies the necessary and sufficient properties of safety and liveness for SMR when players controlling a majority of the consensus votes follow the protocol in every round. We then prove that the Tenderstake protocol is SINCE and fair, which using Theorem 3.5.8, implies Tenderstake achieves ByRa SMR. We conclude in Section 3.8.

3.2 Related Work

There is a growing appreciation that incentivization is not only important, but necessary, to ensure the successful instantiation of an SMR protocol. Many works have argued for the incentivization of players in SMR protocols [63, 98, 94, 10, 9, 99, 34, 92, 42, 17, 73, 65, 83, 16, 3, 75] while many others demonstrate the critical need for incentive compatibility in tokenized SMR protocols [43, 12, 14, 7, 26, 48, 49, 85, 86, 28, 32, 96].

The characterizations of Byzantine and rational, coupled with that of altruistic players who always follow the protocol, segues into the BAR player characterization model as introduced in [6]. We amend the player characterizations to only include those of Byzantine and rational players in what we call the ByRa model, removing any dependency on altruistic players.

A very similar player model is discussed in [83, 75], but with respect to one-shot multiparty computation (MPC). We extend the action space of a one-shot

MPC to allow for indefinite, sequentialized action profiles in line with those of SMR protocols. We introduce the necessity for strict maximization of expected utility to ensure rational players always follow a protocol. This is opposed to [83, 75], where it is claimed that equality of utility will suffice to ensure a rational player will choose one strategy over another, a strong assumption which we prove to be unnecessary. We also allow the adversary to behave arbitrarily, as done in [75].

Although [75] labels this player model as *mixed-behaviour*, and buzzwords such as *Price of Malice* and *Price of Anarchy* are associated with the model in [83], there has been no consensus on the naming of player models containing only Byzantine and rational players. Distributed computing literature since then has been apparently divided on which model to use, while the BAR model remains prevalent. We believe this is, among other possibilities, a consequence of the ambiguity of these names compared to the clarity of BAR. We thus refer to our version of this player model as the ByRa model. The only examples of this player model in SMR literature making meaningful attempts to remove altruistic entities are in [9, 99].

Table 3.1 exhibits the shortcomings of related work in providing SMR protocols that guarantee rational players always follow the protocol (SINCE), and that prevent an adversary from increasing their share of stake to destroy the system (fair). Table 3.1 also includes our proposal, Tenderstake, as a standard against which to compare these works.

In [9], it is implicitly assumed rewards are paid to all players who contribute to consensus on a block. This is non-trivial in the ByRa model, as rewards in their system depend on message delivery. From a protocol's perspective, these messages need to be recorded by a proposer at some point in the protocol, and rational proposers may be incentivized to omit players, as is the case in previous works from subsets of the same authors [11, 12]. We address this omission in the Tenderstake protocol, providing an explicit solution in the ByRa model.

Although [99] provides an SMR protocol which approaches SINCE, they do not provide a rigorous player model excluding altruistic players, and in the presence of a deviating adversary, there are strategies which strictly outperform the recommended protocol strategy for rational players, preventing both strong incentive compatibility and fairness. Such a strategy involves ignoring large forks which per the protocol should be included for rewards, and waiting for these forks to become stale with respect to the longest chain, so as to ensure blocks not in the fork do not get penalized by the fork. Although these forks are likely adversarial, they stand as one of potentially many deviations that a rational player would take.

After the publication of [79] (on which this chapter is based), the Colordag protocol was published [2], which follows a similar approach to [99]. Colordag provides an even stronger notion of incentive compatibility to that of [99] by explicitly removing the ability for an adversary to fork (with arbitrarily high probability). As such, it is feasible that the properties of a Colordag blockchain are as strong as Tenderstake in the Threat Model of Section 3.6.1, with the added benefit of being applicable to the permissionless setting.

A purely economic approach to SMR protocols is taken in [98]. The proposed player model only considers rational players, and depends on a dominating cost for certain deviations that is not quantifiable within the protocol game of maximising stake. In this chapter, we demonstrate that it is possible to construct a protocol, Tenderstake, that strictly maximizes stake by following the protocol. As following the protocol maximizes the value of stake in [98], Tenderstake captures the same maximization of value without the potentially problematic dependency on unquantifiable external costs unrelated to quantity of stake.

One of the legacy works in relation to fairness and incentive compatibility of SMR protocols is Fruitchains [92]. The Fruitchains player model consists of an altruistic majority of players and a cooperative rational minority. Fruitchains crucially relies on an underlying blockchain satisfying an SMR protocol in order to guarantee fairness of rewards. They fail to consider the incentives of all parts of the system, relying on an altruistic majority in order to guarantee the underlying blockchain satisfies the required SMR properties. They then add a section where claims of incentive compatibility for non-cooperative rational players are made. The authors claim a protocol is incentive compatible if fairness of rewards has already been guaranteed. As fairness in their system is only guaranteed if a majority of players follow the protocol, there is no logical implication which proves that rational players will always follow the protocol, required for incentive compatibility. This is insufficient to guarantee SMR in the ByRa model. This dependence on an underlying correct-by-default SMR protocol/trusted third-party is also demonstrated in [34, 69], where claims of incentive compatibility and fairness do not hold in the ByRa model.

3.3 Preliminaries

In our system, an SMR protocol Π consists of n players owning shares of a finite resource, which we will refer to as *stake*, and denoted $Stake^1$ at initialization. Π proceeds in fixed-time periods, which we refer to as *rounds*, beginning in round 1. For any height $\mathcal{H} > 1$ of the blockchain, players participate in Π to decide on a block for that height. Reaching consensus on a block will involve one or more successful protocol steps. After a block has been decided for height $\mathcal{H} \geq 1$, the total stake in the system is denoted $\mathit{Stake}^{\mathcal{H}}$ with player shares of $\mathit{Stake}^{\mathcal{H}}$ denoted $\mathbb{S}_1^{\mathcal{H}}, ..., \mathbb{S}_n^{\mathcal{H}}$. Without loss of generality, we assume $\sum_{i=1}^{n} \mathbb{S}_i^{\mathcal{H}} = 1$, and for all $i \in \{1, ..., n\}$, $\mathcal{H} \geq 1$, $\mathbb{S}_i^{\mathcal{H}} < \frac{1}{2}$.

Definition 3.3.1. *For an SMR protocol Π, the recommended strategy, denoted str_Π, is the strategy that Π requires players to follow in order to successfully achieve SMR.*

3.4 A Game-Theoretic Framework for SMR

In this section we formalize the ByRa framework for SMR protocols, where participants are either adversarially or rationally motivated. This is in response to the existential threat posed by the growing trend of players managing SMR protocols acting in a profit-maximising manner [43] in protocols where security guarantees depend on honest players. Furthermore, this framing is made quite naturally, given SMR protocols are accurately modelled as games with strict incomplete information as defined in Definition 2.2.1.

This is a crucial progression from existing standards in distributed systems literature where some number of non-adversarial players are honest-by-default. Due to the distributed nature of SMR protocols, as a baseline we need to account for some portion of adversarial players who can behave arbitrarily with unknown utility functions. The remaining rational players follow some known utility function, and attempt to choose the actions which maximize it. To ensure the honest behaviour of rational players in SMR protocols within this setting, following the protocol strategy must maximize the expected utility of rational players. We define these player characterizations here formally as the ByRa model.

Definition 3.4.1. *The ByRa model consists of Byzantine and Rational players. A player is:*

- *Byzantine if they deviate arbitrarily from the recommended strategy within a game with unknown utility function. Byzantine players are chosen and controlled by an adversary $\mathcal{A}$.*
- *Rational if they choose uniformly at random from all mixed strategies which maximize their known utility function assuming all other players are rational.*

This definition of rational players omits tie-breaking assumptions that bias a rational player to certain strategies over others with equal utility. If we have a protocol that requires rational players to choose a certain strategy, it is necessary to make the payoff for that strategy strictly greater than that of any other.

A rational player who assumes all other players are rational is known as an *oblivious* rational player [83, 84]. A rational player who is not oblivious knows there are players in the system controlling a non-negligible share of stake, controlled by an adversary, who may try to break safety and liveness. Adding this to the private information of a rational player adds a probability of safety and liveness failing if protocol actions are not followed, which becomes 1 in the presence of a maximal adversary. This outcome has a critical cost for rational players (as used in [9, 98, 83, 84]), which can be made arbitrarily high to prevent rational players from deviating from protocol actions, and the following of protocol actions trivial.

We believe this is highly unrepresentative of rational players in SMR protocols today, particularly in light of the clear recent evidence that miners can and are deviating from protocol actions to increase their on-chain rewards [43]. As such, in the rest of this chapter, we assume all rational players are oblivious, and prove our main lemmas and theorems given this weakest possible assumption.

To consider rational players in any game, it is necessary to explicitly define what their utility functions are. Inkeeping with the tokenized assumptions of our model, we let rational player utility be measured in stake as described by the blockchain. By their nature, tokenized SMR protocols require it to be expensive to deviate from the protocol actions, encouraging honest behaviour through stake rewards, and/or stake punishments for dishonest behaviour. Given the unprecedented levels of SMR protocol usage as a result of tokenization, we see stake as the driving utility measure for the players who participate in these protocols.

As total stake is only meaningful with respect to a particular time-point, and SMR protocols are played indefinitely, rational players will seek to maximize their total stake at all possible rounds sufficiently far into the future. Therefore, when discussing incentivization and player utility, it is necessary to refer to stake/share/total stake with respect to rounds. As we are using the round variable as a counter, and some rounds may be unsuccessful, it cannot be independently used to determine the height, and vice versa. Rather than add notation to relate the two, we treat them separately, and make it clear from context which is being used. When referring to stake/share/total stake with respect to particular rounds, we use superscripts involving r, whereas when discussing these variables with respect to the height of the blockchain, we use superscripts involving $\mathcal{H}$.

Notation 3.4.2. *For an SMR protocol* Π*, we denote by* α *the maximal share of stake such that for players controlling greater than* $1 - \alpha$ *of the stake following the SMR protocol, safety and liveness are achieved. The exact value of* α *will depend on the network distribution assumptions, in line with the results of [45], which must be contained in the threat model.*

For some security parameter $\kappa \in \mathbb{N}$, our goal is to guarantee that SMR can be achieved (that is, both safety and liveness are satisfied) in the ByRa model with probability greater than $1 - \mathit{negl}(\kappa)$ over any polynomial in κ rounds.

We first need to introduce an equivalence relation for mixed strategies over finite rounds. When we state the protocol strategy which needs to be followed to achieve SMR, although there is an infinite number of strategy encodings, we only require players to follow strategies which result in actions as outlined by the protocol. We are indifferent to how this is achieved. If a strategy is encoded differently to the recommended protocol strategy, but results in actions as prescribed by the protocol with probability greater than $1 - \mathit{negl}(\kappa)$ over any polynomial in κ rounds, we see this as equivalent to the recommended protocol strategy.

Definition 3.4.3. *For a player* $\mathcal{P}_i$ *at initialization, and round* $r' \geq 1$*, two mixed strategies* str_i^a *and* str_i^b *are equivalent with respect to round* r' *if for all rounds* r*,* $1 \leq r \leq r'$*, and private informations* $t_i^r \in T_i^r$*, it is the case that* $str_i^a(t_i^r) = str_i^b(t_i^r)$*. We use* $str_i^a \equiv^{r'} str_i^b$ *to denote this equivalence relation. If* $str_i^a \equiv^{r'} str_i^b$ *for all rounds* r' *polynomial in* κ*,* str_i^a *and* str_i^b *are equivalent, denoted by* $str_i^a \equiv str_i^b$*.*

With this equivalence relation, we can now define what it means for a protocol to achieve SMR in the ByRa model. In this chapter, after deciding on a block at height $\mathcal{H} \geq 1$, we denote the adversarial share of stake by $\mathbb{S}_{\mathcal{A}}^{\mathcal{H}}$.

Definition 3.4.4. *For an SMR protocol* Π *and round* r*, let* p_Π^r *be the probability that players controlling more than* $1-\alpha$ *of the total stake follow a mixed strategy* $str \equiv^r str_\Pi$ *up to and including round* r *for any* $\mathbb{S}_{\mathcal{A}}^1 < \alpha$*.* Π *achieves ByRa SMR if for all rounds* r' *polynomial in* κ *it holds that* $p_\Pi^{r'}$ *is greater than* $1 - \mathit{negl}(\kappa)$*. Otherwise,* Π *fails in the ByRa model.*

Towards the goal of achieving ByRa SMR, we need to formally define rational utility as measured in stake. For a rational player $\mathcal{P}_i$ with private information t_i^r and round $r' \geq r$, we have:

$$u_i^{r'}(t_i^r, \mathcal{X}_i^r, ..., \mathcal{X}_i^{r'}, \mathcal{X}_{-i}^r, ..., \mathcal{X}_{-i}^{r'}) = \mathbb{S}_i^{r'} \mathit{Stake}^{r'}. \tag{3.1}$$

However, in a game with strict incomplete information as is the case in an SMR protocol, a rational player $\mathcal{P}_i$ with private information t_i^r will not know their own future private information values (required to choose their actions), the private informations of the other players, or str_{-i}, before choosing str_i. Therefore, $\mathcal{P}_i$ must choose the mixed strategy which maximizes $\mathcal{P}_i$'s expected stake at round r', denoted $E(\mathbb{S}_i^{r'} \mathit{Stake}^{r'})$, according to the probability distribution that $\mathcal{P}_i$ attributes to possible values for these unknowns. This distribution will be contained in t_i^r.

Thus, knowing t_i^r is sufficient to calculate $\mathcal{P}_i$'s expected utility of a particular strategy at round r', which we express mathematically by $E(\mathbb{S}_i^{r'} \mathit{Stake}^{r'} | t_i^r, str_i)$. We state this formally in Definition 3.4.5.

Definition 3.4.5. *For an SMR protocol* Π *and rational player* $\mathcal{P}_i$ *with private information* t_i^r*, mixed strategy* str_i*, and a particular round* $r' \geq r$*, the expected utility of* str_i *for* $\mathcal{P}_i$ *at round* r' *is denoted* $\overline{u}_i^{r'}(t_i^r, str_i)$ *and is described by* $\overline{u}_i^{r'}(t_i^r, str_i) = E(\mathbb{S}_i^{r'} \mathit{Stake}^{r'} | t_i^r, str_i)$.

As such, for a rational $\mathcal{P}_i$ in an SMR protocol Π with private information t_i^r, $\mathcal{P}_i$ will choose the mixed strategy str_i which maximizes $\overline{u}_i^{r'}(t_i^r, str_i)$. To establish the existence, or not, of such a mixed strategy, we introduce an inequality in Definition 3.4.6 which allows us to pairwise rank mixed strategies by expected utility.

Definition 3.4.6. *For an SMR protocol* Π*, rational player* $\mathcal{P}_i$ *and two mixed strategies* str_i^a*,* str_i^b*,* str_i^a *strictly dominates* str_i^b *in expectation if there exists* $r'' \geq r$*,* r'' *polynomial in* κ *, such that for all* $r' > r''$*,* $\overline{u}_i^{r'}(t_i^r, str_i^a) > \overline{u}_i^{r'}(t_i^r, str_i^b)$*. If* str_i^a *strictly dominates* str_i^b *in expectation, we denote this relationship by* $str_i^a >_u str_i^b$*.*

Using the strict dominance in expectancy relationship, we can formally define what we require from an SMR protocol in order for rational players to follow the recommended protocol strategy. This requirement is strong incentive compatibility in expectation, and is defined in Definition 3.4.7.

Definition 3.4.7. *An SMR protocol* Π *is strong incentive compatible in expectation (SINCE) if for any rational player* $\mathcal{P}_i$*,* $str_\Pi >_u str_i$ *for all mixed strategies* $str_i \in Str_i$*, with* Str_i *the set of mixed strategies available to* $\mathcal{P}_i$*, such that* $str_i \not\cong str_\Pi$*.*

For a protocol to be SINCE in the ByRa model ensures that all rational players will follow the recommended protocol strategy. However, SINCE is not on its own sufficient to ensure the safety and liveness of an SMR protocol in ByRa model. It is still possible for an adversary to gain more than their fair share of rewards, and as such, increase their total share above the critical threshold of α. Towards achieving SMR in the ByRa model, it must be ensured that the adversarial share remains strictly bounded by the threshold α required to achieve SMR if all non-adversarial players follow the protocol. We explicitly define what we mean by fairness in the ByRa model in Definition 3.4.8.

Definition 3.4.8. *An SMR protocol* Π *with adversary* $\mathcal{A}$ *is fair in the ByRa model if* $P(\mathbb{S}^r_{\mathcal{A}} \leq \mathbb{S}^1_{\mathcal{A}}) > 1 - \mathit{negl}(\kappa)$ *for any round* $r \geq 1$ *.*

3.5 Achieving SMR in the ByRa Model

With SINCE and fairness, we have two properties which turn out to be crucial in achieving ByRa SMR. Towards our final goal of proving that the properties of SINCE and fairness are necessary, and together sufficient, to achieve ByRa SMR, the first step is to prove in Lemma 3.5.6 that SINCE is necessary. To allow us to prove this result, we introduce notation which allows us to consider, for a potential SMR protocol, the strategies from which rational players choose.

Definition 3.5.1. *For a rational player* $\mathcal{P}_i$ *with a set of mixed strategies* Str_i*, let* $Str_i^{NSD} \subseteq Str_i$ *be such that for all* $str_i \in Str_i^{NSD}$*, there does not exist a* $str_i^u \in Str_i$*, such that* $str_i^u >_u str_i$*.*

That is, if a mixed strategy $str \in Str_i$ is in the set Str_i^{NSD}, there is no strategy for $\mathcal{P}_i$ which strictly dominates str in expectancy. We provide the following Lemmas towards establishing that rational players will choose strategies exclusively from Str_i^{NSD}.

Lemma 3.5.2. *For an SMR protocol* Π*, a rational player* $\mathcal{P}_i$*, any strategy* $str_i^a \in Str_i$ *, and* $|Str_i| \geq 2$*, either* $str_i^a \in Str_i^{NSD}$ *or there is some* $str_i^b \in Str_i^{NSD}$ *such that* $str_i^b >_u str_i^a$*.*

Proof. We will do this by induction over the cardinalities of Str_i. First we check $|Str_i| = 2$. If str_i^a is in Str_i^{NSD}, we are finished. Assume otherwise. That is, $str_i^b >_u str_i^a$, which implies $str_i^a \not>_u str_i^b$, and as such, $str_i^b \in Str_i^{\mathrm{NSD}}$ as required.

Assume the inductive hypothesis for $|Str_i| = k$.

Now, given this assumption, we must prove our hypothesis holds for $|Str_i| = k + 1$. Consider a strategy $str_i^c \in Str_i$. We need to prove either $str_i^c \in Str_i^{\mathrm{NSD}}$, or there exists $str \in Str_i^{\mathrm{NSD}}$ with $str >_u str_i^c$. If str_i^c is not strictly dominated by any strategy $str \in Str_i$, then $str_i^c \in Str_i^{\mathrm{NSD}}$.

Assume instead there exists some strategy $str_i^a \in Str_i$, $str_i^a >_u str_i^c$. Consider $Z_i = Str_i \backslash \{str_i^c\}$. By the inductive assumption, either $str_i^a \in Z_i^{\mathrm{NSD}}$, or there exists $str_i^b \in Z_i^{\mathrm{NSD}}$ such that $str_i^b >_u str_i^a$. If $str_i^a \in Z_i^{\mathrm{NSD}}$, then $str_i^a \in Str_i^{\mathrm{NSD}}$, which implies there exists $str \in Str_i^{\mathrm{NSD}}$ such that $str >_u str_i^c$. Otherwise, if $str_i^a \notin Z_i^{\mathrm{NSD}}$, there exists $str_i^b \in Z_i^{\mathrm{NSD}}$, with $str_i^b >_u str_i^a$. As $str_i^b >_u str_i^a$, and $str_i^a >_u str_i^c$, this implies $str_i^b >_u str_i^c$. As $str_i^b \in Z_i^{\mathrm{NSD}}$, and $str_i^b >_u str_i^c$, this implies $str_i^b \in Str_i^{\mathrm{NSD}}$. Therefore, there exists $str \in Str_i^{\mathrm{NSD}}$ such that $str >_u str_i^c$. □

As rational players choose uniformly at random from all mixed strategies which maximize utility, from Lemma 3.5.2 for a rational player $\mathcal{P}_i$ these mixed strategies will be contained in Str_i^{NSD}. Moreover, Definition 3.4.1 states that $\mathcal{P}_i$ chooses from these mixed strategies in Str_i^{NSD} with uniform probability. Therefore, to ensure rational players follow str_Π with probability at least $1 - \mathit{negl}(\kappa)$, we must identify the conditions where for any rational player $\mathcal{P}_i$, $Str_i^{\mathrm{NSD}} = \{str_\Pi\}$. We state this explicitly in Observation 3.5.3.

Observation 3.5.3. *A rational player $\mathcal{P}_i$ follows str_Π with probability greater than $1 - \mathit{negl}(\kappa)$ if and only if $Str_i^{NSD} = \{str_\Pi\}$.*

The precise conditions where $Str_i^{\mathrm{NSD}} = \{str_\Pi\}$ for a rational player $\mathcal{P}_i$ are identified in Lemma 3.5.4.

Lemma 3.5.4. *For an SMR protocol Π and a rational player $\mathcal{P}_i$, $Str_i^{NSD} = \{str_\Pi\}$ if and only if Π is strong incentive compatible in expectation.*

Proof. If an SMR protocol Π is SINCE, then for any rational player $\mathcal{P}_i$, str_Π strictly dominates all other strategies in expectation. From Lemma 3.5.2, this implies $Str_i^{\mathrm{NSD}} = \{str_\Pi\}$.

Now we need to show if $Str_i^{\mathrm{NSD}} = \{str_\Pi\}$, then Π is SINCE. From Lemma 3.5.2, we know for any strategy str_i^a, either $str_i^a \in Str_i^{\mathrm{NSD}}$ or there is some $str_i^b \in Str_i^{\mathrm{NSD}}$ such that $str_i^b >_u str_i^a$. As the only strategy in Str_i^{NSD} is str_Π, this implies for any strategy $str_i^a \not\equiv str_\Pi$, $str_\Pi >_u str_i^a$. This implies Π is SINCE, as required. □

Corollary 3.5.5. *For an SMR protocol Π and a rational player $\mathcal{P}_i$, $P(\mathcal{P}_i$ chooses $str_\Pi) > 1 - \mathit{negl}(\kappa)$ if and only if Π is strong incentive compatible in expectation.*

Proof. Follows from Observation 3.5.3 and Lemma 3.5.4. □

This allows us to prove SINCE is a necessary property to achieve ByRa SMR.

Lemma 3.5.6. *For an SMR protocol* Π*, if* Π *is not strong incentive compatible in expectation, then* Π *fails in the ByRa model.*

Proof. Consider such a protocol Π. As a consequence of not SINCE, for a rational player $\mathcal{P}_i$, this means $P(\mathcal{P}_i \text{ chooses } str_\Pi)$ is not greater than $1 - \mathit{negl}(\kappa)$, applying Corollary 3.5.5. From Definition 3.4.4 we are required to consider $\mathbb{S}^1_\mathcal{A}$ maximal. Given this rational $\mathcal{P}_i$ and a maximal adversary, there is now players controlling greater than or equal to α of the total stake who will not choose a strategy equivalent to str_Π with non-negligible probability in κ. Using the notation of Definition 3.4.4, this means p^r_Π is not greater than $1 - \mathit{negl}(\kappa)$ for some $r \geq 1$, which implies Π fails in the ByRa model. $\square$

Using similar arguments, we are able to prove fairness is also necessary for a protocol to achieve ByRa SMR.

Lemma 3.5.7. *For an SMR protocol* Π*, if* Π *is not fair then* Π *fails in the ByRa model.*

Proof. If Π is not fair, there exists $r \geq 1$ such that $P(\mathbb{S}^r_\mathcal{A} > \mathbb{S}^1_\mathcal{A})$ is not negligible in κ. From Definition 3.4.4, we are required to consider the case where $\mathbb{S}^1_\mathcal{A}$ is maximal. In this case, the probability that the adversary controls greater than or equal to α of the stake at round r is is non-negligible in κ given $P(\mathbb{S}^r_\mathcal{A} > \mathbb{S}^1_\mathcal{A})$ is non-negligible in κ. Given the uniform strategy selection probability of Byzantine players across all possible strategies, this implies that p^r_Π is not greater than $1 - \mathit{negl}(\kappa)$. Therefore, Π fails in the ByRa model. $\square$

Collecting the results of this section, with some additional proof-work, we are equipped to prove the main theorem of the chapter, Theorem 3.5.8.

Theorem 3.5.8. *For an SMR protocol* Π*,* Π *achieves ByRa SMR if and only if* Π *is strong incentive compatible in expectation and fair.*

Proof. For an SMR protocol Π, we will first prove that if Π achieves ByRa SMR then Π is SINCE and fair. Using the contrapositive of Lemma 3.5.6, we have that if Π achieves ByRa SMR (does not fail in the ByRa model), then Π is SINCE. Similarly, using the contrapositive of Lemma 3.5.7, we have that if Π achieves ByRa SMR, then Π is fair.

We now need to prove if Π is SINCE and fair then Π achieves ByRa SMR. By SINCE and Corollary 3.5.5, this implies all rational players will always choose str_Π. Furthermore, as Π is fair, from Definition 3.4.8, we know rational players will maintain greater than $1 - \alpha$ of the stake in every round with probability greater than $1 - \mathit{negl}(\kappa)$. Therefore, we have players controlling greater than $1-\alpha$ of the stake who will follow str_{Π} with probability greater than $1 - \mathit{negl}(\kappa)$, which is precisely the definition of Π achieving ByRa SMR from Definition 3.4.4. □

This important theorem completes the first part of the chapter, identifying the properties of SINCE and fairness as both necessary, and together sufficient, for a protocol to achieve ByRa SMR, independently of network assumptions and adversarial capabilities.

3.6 Tenderstake

In this section, we provide the encoding of Tenderstake, and give an overview of the main differences between the Tenderstake protocol and Tendermint. We assume a partially synchronous network communication model as in Tendermint [68]. Players are connected to nodes in a dynamic wide area network, with each node having direct connections to a subset of all other nodes, forming a sparsely connected graph of communication channels between nodes. Non-Byzantine player messages are transmitted through *gossiping*; players send a message to neighbouring nodes, who echo messages to their neighbours until all nodes eventually receive the message. Formally, there is global stabilization round $GSR > 0$, such that all messages sent at round $r_{send} > 0$ are delivered by round $r_{deliver} = max(r_{send}, GSR) + \Delta$ for some unknown number of rounds $\Delta > 0$.

We assume rational players are aware that there is a fixed, but unknown, upperbound Δ on message delivery between players in synchrony, which we refer to as Δ-synchrony, but are unaware of how many players are in Δ-synchrony at any given time. For a message m, a call to **broadcast**(m) sends m to all players, including oneself, under the same gossiping specification. This is partial synchrony as defined in [45].

3.6.1 Threat Model

In Tenderstake, protocol actions take negligible amounts of time compared to network delays, so if all non-Byzantine players behave correctly and receive the same sequence of messages their machines will be in the same state. Rational players ignore messages which have not been signed using a protocol-associated private key. We do not consider coalitions of rational players.

We consider an adversary $\mathcal{A}$ with the following properties:

1. $\mathcal{A}$ can read all messages sent by non-Byzantine parties, but cannot existentially forge signatures.

2. $\mathcal{A}$ can control and coordinate all Byzantine players in any way, with unknown utility function.

3. At initialization we have $\frac{1}{3} - \delta < \mathbb{S}^1_{\mathcal{A}} < \frac{1}{3} = \alpha$, for some $\delta > 0$, in line with the partially synchronous network distribution limits [45].

4. At initialization, $\mathcal{A}$ can choose to corrupt any $1 \leq f < n - 2$ players, say $\mathcal{P}_1, ..., \mathcal{P}_f$ with shares $\mathbb{S}^1_1, ..., \mathbb{S}^1_f$, such that $\sum_{i=1}^{f} \mathbb{S}^1_i = \mathbb{S}^1_{\mathcal{A}}$.

5. Given $\mathcal{A}$ corrupts players $\mathcal{P}_1, ..., \mathcal{P}_f$ as Byzantine for consensus on a block at height $\mathcal{H}$ with shares $\mathbb{S}^{\mathcal{H}-1}_1, ..., \mathbb{S}^{\mathcal{H}-1}_f$, the adversarial share at the proceeding height is calculated as $\mathbb{S}^{\mathcal{H}}_{\mathcal{A}} = \sum_{i=1}^{f} \mathbb{S}^{\mathcal{H}}_i$.

We focus on static adversaries for simplicity, and leave other adversarial types, such as adaptive or mobile[104], as future work. The added complexity of such adversaries only stands to detract from the primary goals of the chapter, that is, to demonstrate the importance of ByRa SMR and how it can be achieved with Tenderstake. We also choose not to allow players to join/leave the protocol for the same reasons; a desire to limit complexity and to maintain weak network communication model assumptions. We discuss the addressing of these decisions as part of future work in Section 3.8.

3.6.2 Protocol Outline

We now describe the pseudocode of Tenderstake as outlined in Algorithm 1. As the goal of Section 3.6 is to amend Tendermint to achieve ByRa SMR, readers of [31] will notice that we use large parts of the code and descriptions from that work. We describe the entire code here for completeness, and highlight the differences in Tenderstake to Tendermint as they arise. The two fundamental additions to the Tendermint protocol used by Tenderstake are proof-of-transition and slashing functionalities, described in detail at the end of this section, and included in the code of Algorithm 1. Proof-of-transition ensures players who send a message at a particular height/ epoch/ step have gotten there by following the protocol, while the slashing functionality enforces the use of proofs-of-transition, as well as the sending of valid messages in general, by punishing players for sending invalid messages.

Algorithm 1 Tenderstake protocol for a player $\mathcal{P}_i$

```
1: function $Initialize(Genesis)$
2:     $Blockchain_i := [Genesis]$                          ▷ $\mathcal{P}_i$'s blockchain as a vector
3:     $\mathcal{H}_i := 1$                                 ▷ Tracks height of $Blockchain_i$
4:     $epoch_i := 1$
5:     $step_i \in \{propose, prevote, precommit\}$
6:     $lockValue_i := nil$
7:     $lockEpoch_i := -1$
8:     $validValue_i := nil$
9:     $validEpoch_i := -1$
10:    $Stake_i := Genesis.stake()$                         ▷ Total stake
11:    $Shares_i := Genesis.shares()$                       ▷ Vector of player shares
12:    $Reward_i := Genesis.reward()$                       ▷ Per-Block reward
13:    $DeviationProofs_i := [nil\ for\ j \in \{1, ..., n\}]$  ▷ Deviation proofs
14:    $prevoteProof_i := nil$
15:    $precommitProof_i := nil$
16: upon start do $StartEpoch(1)$
17: function $StartEpoch(epoch)$
18:    $epoch_i \leftarrow epoch$
19:    $step_i \leftarrow propose$
20:    if proposer$(\mathcal{H}_i, epoch_i) = \mathcal{P}_i$ then
21:        if $validValue_i \neq nil$ then
22:            $proposal_i \leftarrow validValue_i$
23:        else
24:            $proposal_i \leftarrow getValue().include(DeviationProofs_i)$
25:        broadcast⟨PROPOSAL, $\mathcal{H}_i, epoch_i, proposal_i, validEpoch_i, prevoteProof_i$⟩
26:    else
27:        schedule $OnTimeoutPropose(\mathcal{H}_i, epoch_i)$ to be executed after $timeout()$

28: upon ⟨PROPOSAL, $\mathcal{H}_i, epoch_i, value, -1, proof$⟩ with $valid(proof)$ from proposer$(\mathcal{H}_i, epoch_i)$
    while $step_i = propose$ do
29:    if $valid(value) \wedge (lockEpoch_i = -1 \vee lockValue_i = value)$ then
30:        broadcast ⟨PREVOTE, $\mathcal{H}_i, epoch_i, value, prevoteProof_i$⟩
31:    else
32:        broadcast ⟨PREVOTE, $\mathcal{H}_i, epoch_i, nil, prevoteProof_i$⟩
33:    $step_i \leftarrow prevote$

34: upon ⟨ PROPOSAL, $\mathcal{H}_i, epoch_i, value, validEpoch, proofProposal$⟩ with $valid(proofProposal)$
    from proposer$(\mathcal{H}_i, epoch_i) \wedge > \frac{2}{3}$ ⟨PREVOTE, $\mathcal{H}_i, validEpoch, value, proofPrevote$⟩ with
    $valid(proofPrevote)$ while $\big(step_i = propose \wedge (validEpoch \geq 0 \wedge validEpoch < epoch_i)\big)$ do
35:    $prevoteProof_i \leftarrow proof(> \frac{2}{3}$ ⟨PREVOTE, $\mathcal{H}_i, validEpoch, value$⟩ $\cup\ prevoteProof_i)$
36:    if $valid(value) \wedge (lockEpoch_i < validEpoch \vee lockValue_i = value)$ then
37:        broadcast ⟨PREVOTE, $\mathcal{H}_i, epoch_i, value, prevoteProof_i$⟩
38:    else
39:        broadcast ⟨PREVOTE, $\mathcal{H}_i, epoch_i, nil, prevoteProof_i$⟩
40:    $step_i \leftarrow prevote$

41: upon $> \frac{2}{3}$ ⟨PREVOTE, $\mathcal{H}_i, epoch_i, *, proof$⟩ with $valid(proof)$ while $step_i = prevote$ for the
    first time do
42:    $precommitProof_i \leftarrow proof(> \frac{2}{3}$⟨PREVOTE, $\mathcal{H}_i, epoch_i, *$⟩$)$
43:    schedule $OnTimeoutPrevote(\mathcal{H}_i, epoch_i)$ to be executed after $timeout()$
```

Algorithm 1 Tenderstake protocol (ctd.)

```
44: upon $\langle \mathsf{PROPOSAL}, \mathcal{H}_i, epoch_i, value, *, proofProposal \rangle$ with $valid(proofProposal)$ from
    $\mathsf{proposer}(\mathcal{H}_i, epoch_i) \wedge > \frac{2}{3} \langle \mathsf{PREVOTE}, \mathcal{H}_i, epoch_i, value, proofPrevote \rangle$
    with $valid(proofPrevote)$ while $valid(value) \wedge step_i \geq prevote$ for the first time do
45:     if $step_i = prevote$ then
46:         $lockValue_i \leftarrow value$
47:         $lockEpoch_i \leftarrow epoch_i$
48:         $precommitProof_i \leftarrow proof(> \frac{2}{3} \langle \mathsf{PREVOTE}, \mathcal{H}_i, epoch_i, value \rangle)$
49:         broadcast $\langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch_i, value, precommitProof_i \rangle$
50:         $step_i \leftarrow precommit$
51:     $validValue_i \leftarrow value$
52:     $validEpoch_i \leftarrow epoch_i$

53: upon $> \frac{2}{3} \langle \mathsf{PREVOTE}, \mathcal{H}_i, epoch_i, nil, proof \rangle$ with $valid(proof)$ while $step_i = prevote$ do
54:     $precommitProof_i \leftarrow proof(> \frac{2}{3} \langle \mathsf{PREVOTE}, \mathcal{H}_i, epoch_i, nil \rangle)$
55:     broadcast $\langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch_i, nil, precommitProof_i \rangle$
56:     $step_i \leftarrow precommit$

57: upon $> \frac{2}{3} \langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch_i, *, proof \rangle$ with $valid(proof)$ for the first time do
58:     $prevoteProof_i \leftarrow proof(> \frac{2}{3} \langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch_i, * \rangle)$
59:     schedule $OnTimeoutPrecommit(\mathcal{H}_i, epoch_i)$ to be executed after $timeout()$
```

In Tenderstake, every correct player is initialized by passing a block *Genesis* to the *Initialize* function (line 1). This ensures all players start from a common state. Block *Genesis* contains information on player shares, stake and per-block reward at initialization.

The algorithm is presented as a set of **upon** rules that are to be executed automatically once the corresponding logical condition is *TRUE*. Variables with sub-index i denote player $\mathcal{P}_i$'s local state variables, while those without are value placeholders. The sign $*$ denotes any value. We use the convention of $> \frac{x}{3}\ m$ **with** *COND* to stand for the logical statement which is *TRUE* if and only if players controlling more than $\frac{x}{3}$ of the total stake with respect to $\mathcal{P}_i$'s blockchain $Blockchain_i$ (represented as a vector in line 2) deliver messages, with each message m satisfying the logical condition *COND*. If m contains a proposed deviator, that deviator's share does not count towards the tally (it would be irregular that a player would affirm a message which tried to destroy their own stake).

As the total voting power in the system is 1, this means if there are new deviators proposed in a particular valid value (line 24) with total share $\mathbb{S}_{dev}$, the maximum total voting share for that value is $1 - \mathbb{S}_{dev}$. This ensures any player $\mathcal{P}_i$ at height $\mathcal{H}$ with share $\mathbb{S}_i^{\mathcal{H}-1}$ after deciding on the block at height $\mathcal{H} - 1$ can only have 0 or $\mathbb{S}_i^{\mathcal{H}-1}$ voting power. Rules ending with 'for the first time' should only be executed on the first time the corresponding condition is *TRUE*.

Algorithm 1 Tenderstake protocol (ctd.)

```
60: upon $> \frac{2}{3}$ $\langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch_i, nil, proof\rangle$ with $valid(proof)$ while $step_i = precommit$
    do
61:     $StartEpoch(epoch_i + 1)$

62: upon $\langle \mathsf{PROPOSAL}, \mathcal{H}_i, epoch, value, *, proofProposal\rangle$ with $valid(proofProposal)$
    from $\mathsf{proposer}(\mathcal{H}_i, epoch) \wedge > \frac{2}{3}$ $\langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch, value, proofPrecommit\rangle$
    with $valid(proofPrecommit)$ do
63:     $newDeviators \leftarrow value.deviators() \setminus Blockchain_i.deviators()$
64:     if $valid(value)$ then
65:         if $|newDeviators| > 0$ then          ▷ True if deviators in $value$ not in $Blockchain_i$
66:             $adjustForSlashing(newDeviators)$
67:         $prevoteProof_i \leftarrow proof(> \frac{2}{3}\langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch, value\rangle)$
68:         $Blockchain_i.append(value.include(prevoteProof_i))$
69:         $Stake_i \leftarrow Stake_i + Reward_i$
70:         $\mathcal{H}_i \leftarrow \mathcal{H}_i + 1$
71:         reset $lockEpoch_i$, $lockValue_i$, $validEpoch_i$, $validValue_i$ to initial values
72:         $StartEpoch(1)$

73: upon $> \frac{1}{3}$ $\langle *, \mathcal{H}_i, epoch, *, proof\rangle$ with $(epoch > epoch_i \wedge valid(proof))$ do
74:     $prevoteProof_i \leftarrow proof(> \frac{1}{3} \langle *, \mathcal{H}_i, epoch, *, *\rangle)$
75:     $StartEpoch(epoch)$

76: function $OnTimeoutPropose(height, epoch)$
77:     if $height = \mathcal{H}_i \wedge epoch = epoch_i \wedge step_i = propose$ then
78:         broadcast $\langle \mathsf{PREVOTE}, \mathcal{H}_i, epoch_i, nil, prevoteProof_i\rangle$
79:         $step_i \leftarrow prevote$

80: function $OnTimeoutPrevote(height, epoch)$
81:     if $height = \mathcal{H}_i \wedge epoch = epoch_i \wedge step_i = prevote$ then
82:         broadcast $\langle \mathsf{PRECOMMIT}, \mathcal{H}_i, epoch_i, nil, precommitProof_i\rangle$
83:         $step_i \leftarrow precommit$

84: function $OnTimeoutPrecommit(height, epoch)$
85:     if $height = \mathcal{H}_i \wedge epoch = epoch_i$ then
86:         $StartEpoch(epoch_i + 1)$

87: upon $m$ from $\mathcal{P}^j$ with $valid(m) = FALSE$ for the first time do
88:     $DeviationProofs_i[j] \leftarrow proof(valid(m) = FALSE)$
89:     broadcast $\langle \mathsf{SLASH}, \mathcal{P}^j, \mathcal{H}_i, epoch_i, m, DeviationProofs_i[j]\rangle$

90: upon $\langle \mathsf{SLASH}, \mathcal{P}^j, m, proof\rangle$ from $\mathcal{P}^k$ with $valid(proof)$ do
91:     if $DeviationProofs_i[j] = nil$ then
92:         $DeviationProofs_i[j] \leftarrow proof$
93:         broadcast $\langle \mathsf{SLASH}, \mathcal{P}^j, m, DeviationProofs_i[j]\rangle$

94: function $adjustForSlashing(newDeviators)$
95:     $slashedShare \leftarrow sum(Shares_i[newDeviators])$
96:     $Shares_i[newDeviators] \leftarrow 0$
97:     $Shares_i \leftarrow [\frac{Shares_i[k]}{1-slashedShare}$ for $k \in [1, ..., n]]$
98:     $Reward_i \leftarrow (1 - slashedShare)Reward_i$
99:     $Stake_i \leftarrow (1 - slashedShare)Stake_i$          ▷ Remove deviating stake
100:    $Stake_i \leftarrow Stake_i + sum([Genesis.shares()[j]$ for $j \in newDeviators]) \cdot Reward_i$     ▷ Slash
    Bonus, not dependant on ordering
```

The algorithm proceeds in epochs, with each epoch having a dedicated proposer. The mapping of epochs to proposers is known to all players, with the function $\mathsf{proposer}(h,\ \mathit{epoch})$ returning the proposer for epoch *epoch* given current blockchain height h. Player state transitions are triggered by message reception and by expiration of the timeout function $\mathit{timeout}()$. Timeouts are to be called once per step during each epoch, and only trigger a transition if the player has not updated their step or epoch variable since starting the timeout function.

In [31] it is proved that non-Byzantine players need to incorporate increasing timeouts in the number of epochs at a particular height to guarantee eventual progression. In Tenderstake, we also incorporate increasing timeouts in epochs, but instead leave the precise definition of the timeout function $\mathit{timeout}()$ to each player. We do however place the following restriction on the $\mathit{timeout}()$ calculation: the value of $\mathit{timeout}()$ is increasing in the number of epochs at every height, such that $\lim\limits_{\mathit{epoch}\to\infty} \mathit{timeout}(\mathit{epoch}) \to \infty$.

The intuition behind this choice is leaving it sufficiently general so as to not risk choosing some specific delta/ function for delta which would expose us to unnecessary optimization analysis, while also ensuring Tenderstake retains the property of increasing timeouts in the number of epochs at each height required in the original Tendermint protocol to guarantee safety and liveness.

Messages in Tenderstake contain one of the following tags:
PROPOSAL, PREVOTE, PRECOMMIT, or SLASH. The PROPOSAL tag is used by the proposer of the current epoch to suggest a potential decision value (line 25), while PREVOTE and PRECOMMIT are votes for a proposed value, as in Tendermint. SLASH messages identify player deviations, and are described in detail at the end of the this section.

Every player $\mathcal{P}_i$ locally stores the following variables in the Tenderstake protocol: step_i, $\mathit{lockValue}_i$, $\mathit{lockEpoch}_i$, $\mathit{validValue}_i$, $\mathit{validEpoch}_i$, Stake_i, Shares_i, Reward_i, and $\mathit{DeviationProofs}_i$, initialized in lines 5-13. The step_i tracks the current step of the protocol execution during the current epoch. The $\mathit{lockValue}_i$ stores the most recent value for which a PRECOMMIT message was sent by $\mathcal{P}_i$ for a non-*nil* value, with $\mathit{lockEpoch}_i$ the epoch in which $\mathit{lockValue}_i$ was updated. As $\mathcal{P}_i$ can only decide on a value *value* if more than $\frac{2}{3}$ voting power equivalent PRECOMMIT messages are received for *value*, possible decision values can be any value locked by more than $\frac{1}{3}$ voting power equivalent players. Therefore any value *value* for which PROPOSAL and more than $\frac{2}{3}$ voting power equivalent PREVOTE messages are received in some epoch is a possible decision value. The $\mathit{validValue}_i$ stores this value, while $\mathit{validEpoch}_i$ stores the epoch where this update occurred. The Stake_i tracks the total stake in the system, and Shares_i the current player shares of Stake_i. The Reward_i is the total reward to be distributed among all players for deciding on the next value in $Blockchain_i$. The $DeviationProofs_i$ vector tracks locally observed deviators as identified by SLASH messages.

Proof-of-Transition Functionality

In Tenderstake, every PROPOSAL, PREVOTE and PRECOMMIT message must be accompanied by a *proof-of-transition* which evidences the transition to the current step claimed by each player is valid. These proofs are stored in the local player variables $prevoteProof_i$ and $precommitProof_i$, with $prevoteProof_i$ also acting as proof for PROPOSAL messages when a player is selected as proposer. For example, in line 57, each PRECOMMIT message must be accompanied by a *proof* which shows that the respective players were at a protocol step which allowed them to send a PRECOMMIT message (lines 49, 55 or 82). This would be true either if the player received PREVOTE messages correctly satisfying the condition at line 53, both of the conditions at lines 44, 45, or the condition at 80 . As these conditions have specific PREVOTE message reception rules, and given there is only one valid PRECOMMIT messages in each case, *valid*(*proof*) is true if and only if the message was generated correctly, i.e. by receiving a set of PREVOTE messages which would trigger that PRECOMMIT message according to the protocol.

Slashing Functionality

If any messages/ proofs are not valid in Tenderstake, players trigger the *Slashing functionality* and send a SLASH message (line 89), which contains a proof that the offending message was indeed invalid (described in detail in Section 3.6.2). SLASH messages, along with proofs-of-transition, are a key addition to Tenderstake in order to prove ByRa SMR, as players identified as deviating by a correct player through a SLASH message will eventually be seen by all correct players. When a player is identified as deviating, their proof-of-deviation is added to the local *DeviationProofs* vector of the observing player. Then when a player is selected as proposer, and $validValue_i = nil$, they add all deviation proofs not already identified in $Blockchain_i$ to their proposed value (line 24). After being seen by all correct players, any deviator will eventually be added to a correct player's proposed value and removed from the protocol through the *adjustForSlashing* function (line 94).

The *adjustForSlashing* function takes as input new decided deviators, deletes their stakes (lines 96, 99), adjusts the remaining player shares to sum to 1 (line

97), recalibrates the per-block reward to keep the per player reward constant throughout a Tenderstake instance (line 98), and distributes the initial reward *Genesis.reward*() times the initial shares of the deviating players among the remaining players in proportion to their stake (line 100).

Life-Cycle of an Epoch

Every epoch starts by a proposer suggesting a value in a Tenderstake message (line 25). If $validValue_i = nil$, this proposed value is generated by the external *getValue*() function (line 24), as in Tendermint. In Tenderstake, players also include in their newly generated propose values any deviation proofs they have received that are not currently in $Blockchain_i$. Otherwise if $validValue_i \neq nil$, the proposer proposes $validValue_i$. The proposer attaches $validEpoch_i$ to the message so other processes are informed of the last epoch in which the proposer observed $validValue_i$ as a possible decision value.

Upon receiving a valid $\langle \mathsf{PROPOSAL}, \mathcal{H}_i$, $epoch_i$, $value$, $validEpoch$, $proofProposal\rangle$ message, a correct player $\mathcal{P}_i$ accepts the proposed value *value* if both the external function *valid*(*value*) returns *TRUE* and either $\mathcal{P}_i$ has not locked any value ($lockEpoch_i = -1$) or $\mathcal{P}_i$ has locked on *value* (line 29). For a valid proposed value *value* with $validEpoch \geq 0$, if $validEpoch > validEpoch_i$ (the proposed value was more recent than $\mathcal{P}_i$'s locked value) or $lockValue_i = value$, $\mathcal{P}_i$ will accept *value* (line 36). Otherwise, $\mathcal{P}_i$ rejects the proposal by sending a PREVOTE message for *nil*. $\mathcal{P}_i$ will also send a PREVOTE message for *nil* if the timeout triggered in line 27 expires and they have not sent a PREVOTE message for any other value during this epoch yet (line 78).

If a correct player $\mathcal{P}_i$ receives a PROPOSAL message for a valid value *value* and PREVOTE messages for *value* from players controlling more than $\frac{2}{3}$ of the share as described by *value*, then it sends a PRECOMMIT message for *value*. Otherwise, they send a PRECOMMIT message for *nil*. A correct process will also send a PRECOMMIT message for *nil* if the timeout triggered in line 43 expires and they have not sent a PRECOMMIT message for their current epoch yet (line 82). A correct player decides on a value *value* if it receives in some epoch *epoch* a PROPOSAL message for *value* and PRECOMMIT messages for *value* from players controlling more than $\frac{2}{3}$ of the share as described by *value*. On a decision, *value*, including proof of the PRECOMMIT messages allowing $\mathcal{P}_i$ to decide on *value*, are appended to $Blockchain_i$ (line 68). Otherwise, to ensure progression, if the timeout triggered at line 59 expires, the player proceeds to the next epoch (line 86).

Proof-of-Deviation

Crucial to the slashing functionality are the *proofs-of-deviation* which can be generated upon the reception of any invalid message. As invalid messages can take various forms, we explicitly define each form of invalid message and how to generate the corresponding proof-of-deviation. Invalid messages in Tenderstake can (1) contradict another message from the same sender, (2) propose invalid values, (3) contain an invalid proof-of-deviation or (4) contain an invalid proof-of-transition.

Any message which does not contain one or more of the deviations outlined in this section is *valid.* As we require a valid, non-contradictory proof-of-transition with every signed non proof-of-deviation message, there is only a small finite combination of valid $step_i, \mathcal{H}_i$, $epoch_i$ and proposed/locked values that will be valid with respect to such a proof-of-transition. As such, the deviations outlined in this section are exhaustive.

We do not explicitly encode proofs-of-deviation in this chapter. Upon a precise specification of Tenderstake, message validity will be verifiable using a finite set of rules, and therefore any messages not following these rules can be provided as proof-of-deviation. However, we now describe the maximum amount of information necessary to prove that a player has deviated, which can then be represented in some, possibly condensed form within a SLASH message to prove a player has deviated. In the following we assume a player $\mathcal{P}_i$ performs the corresponding deviation.

1. Contradictory messages: If a player sends two messages m and m' such that they both contain valid proofs-of-transition, but it is not possible to transition from either of the messages to the other, these messages together constitute a proof of deviation. For example, if there are two messages m and m' from $\mathcal{P}_i$ with the same $\mathcal{H}_i$, $epoch_i$ and $step_i$ tags, or if $\mathcal{P}_i$ proposes a newly generated $getValue()$ after sending a PRECOMMIT message in a preceeding epoch at the same height for a different value (which would mean $validValue_i \neq nil$).

2. Invalid proposed values: As the blockchain value validity predicate is shared by all parties and known a priori, any message from $\mathcal{P}_i$ containing an invalid proposed value *value* can be used as a proof of deviation.

3. Invalid slashing: A slash message is invalid if the accompanying proof-of-deviation is not valid. If the proof-of-deviation is not valid, the corresponding slash message/ proposed value (if it first appears in a proposed value) stands as a proof of deviation.

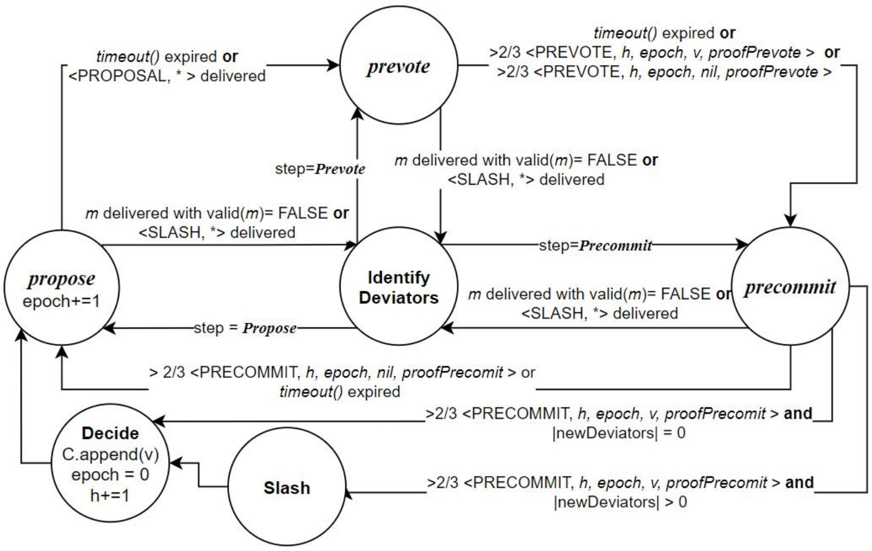
prevote
propose
epoch+=1
Identify Deviators
precommit
Decide
C.append(v)
epoch = 0
h+=1
Slash
timeout() expired or <PROPOSAL, * > delivered
timeout() expired or >2/3 <PREVOTE, h, epoch, v, proofPrevote > or >2/3 <PREVOTE, h, epoch, nil, proofPrevote >
step=Prevote
m delivered with valid(m)= FALSE or <SLASH, *> delivered
m delivered with valid(m)= FALSE or <SLASH, *> delivered
step=Precommit
m delivered with valid(m)= FALSE or <SLASH, *> delivered
step = Propose
> 2/3 <PRECOMMIT, h, epoch, nil, proofPrecomit > or timeout() expired
>2/3 <PRECOMMIT, h, epoch, v, proofPrecomit > and |newDeviators| = 0
>2/3 <PRECOMMIT, h, epoch, v, proofPrecomit > and |newDeviators| > 0

Figure 3.1: A state diagram representation of Tenderstake

4. Invalid proof-of-transition: If a message m is received from a player $\mathcal{P}_i$, where $\mathcal{P}_i$ has not attached (a proof of) messages with tag, height, epoch and value variables which validly trigger the logical conditions necessary to send m, this constitutes an invalid proof-of-transition. These messages, or lack thereof, constitute a proof-of-deviation. As $\mathcal{P}_i$ signs m, the contents of m can be verified, and as such $\mathcal{P}_i$'s attempted proof-of-transition can be proved to belong to $\mathcal{P}_i$ (as the signature must correspond to m and the contained proof-of-transition), and proved to be invalid by all players. Any message sent by $\mathcal{P}_i$ which does not adhere to one of the protocol-specified **broadcast** formats[6] is considered to contain an invalid proof-of-transition. This is because there is no protocol-specified transition that would create such a message.

[6]Can be thought of as junk, but also includes attempted communication between players such as, perhaps, to coordinate collusion.

3.7 Proving Tenderstake achieves ByRa SMR

In this section we prove that Tenderstake achieves SMR in the ByRa model. To this end, we first prove that it is an SMR protocol when more than $\frac{2}{3}$ of the share is controlled by honest players at all times.

Lemma 3.7.1. *It is not possible to generate a valid deviation proof for an honest player.*

Proof. A valid deviation proof for some player $\mathcal{P}_i$ must identify a message from $\mathcal{P}_i$ that is invalid according to one of the methods listed in the Proof-of-Deviation subsection of Section 3.6.2, which covers all possible message deviations. By definition, honest players follow all protocol rules and only send valid messages. We also know that honest player messages cannot be forged under our threat model assumption regarding unforgeable signatures from Section 3.6.1. Furthermore, as $\mathcal{A}$ is static, every message signed by a currently honest player must have been generated honestly (by that same player) at some point in the protocol. Therefore, all honest player messages are valid, and as such, no valid proof-of-deviation described in Section 3.6.2 can be generated for honest players. □

Lemma 3.7.2. *Tenderstake achieves SMR when players controlling more than $\frac{2}{3}$ of the stake are honest.*

Proof. The aim of this proof is to demonstrate that proposed values (line 24) satisfy safety and liveness in Tenderstake. An initialization of Tenderstake is equivalent to a standard Tendermint initialization, in addition to the proof-of-transition and slash functionalities as described in the respective subsections of Section 3.6.2. For deciding on a value at a particular height $\mathcal{H} > 1$, voting share is described by the value decided at height $\mathcal{H} - 1$, or the current proposed value (line 24) if it is valid and contains newly identified deviators. From Lemma 3.7.1, we know no valid proof of deviation can be generated for an honest player. Therefore, honest players can never be included as prospective deviators in valid proposed values in line 24. This ensures that any valid value proposed will maintain honest voting share of more than $\frac{2}{3}$.

Proof-of-transition values are simply additional pieces of information attached to standard Tendermint messages. Identically to Tendermint, Tenderstake does not consider invalid messages for any of the steps needed to decide on a block (lines 29, 44, 62, 34). Consider an epoch during synchrony, with timeouts larger than the message delivery delta Δ for all honest players (necessary to ensure liveness, as in Tendermint) and an honest proposer. This epoch occurs eventually at every height (if no decision has been reached in earlier epochs) as the network communication model and proposer rotation are identical to Tendermint, and the timeout function is increasing and unbounded in the number of epochs. As more than $\frac{2}{3}$ of the voting share is controlled by honest players at all times, honest players will decide on the proposed value during that epoch. This holds for any height $\mathcal{H} > 1$, and the safety and liveness of Tenderstake follows. □

With Tenderstake as an SMR protocol under an honest majority, we now need to prove Tenderstake achieves ByRa SMR. To do this, we will prove that Tenderstake is SINCE and fair in the ByRa model, and apply Theorem 3.5.8. To prove SINCE, we first need some results that bound the reward a player can achieve for deciding on a value. As each decided value requires an accompanying $> \frac{2}{3}$ PRECOMMIT messages to be valid, the maximum amount of values a player can attempt to decide on at once is 1, as the proceeding value will need to point to the decided value and the value's $> \frac{2}{3}$ PRECOMMIT messages. By bounding the reward a player can get for deciding on values and deviators (the only rewarding actions) sufficiently low, we are able to prove that this reward is negligible compared to the potential punishment for being caught, thus preventing rational players from sending invalid messages.

Lemma 3.7.3. *In any instance of Tenderstake, $\mathcal{P}_i$ receives less than $\mathbb{S}_i^1$ Genesis.reward() in total for identifying deviators from the adjustForSlashing function.*

Proof. We can see that the rewards for deciding on new deviators at height $\mathcal{H}$ are distributed at line 100. We will prove that the sum of the rewards distributed by calling line 100 throughout an instance of Tenderstake are less than $\mathbb{S}_i^1 \mathit{Genesis.reward}()$.

Let there be a set of players $\mathit{newDeviators}^{\mathcal{H}}$ controlling *slashedShare* at height $\mathcal{H} - 1$ identified as deviating for the first time in the value at height $\mathcal{H}$. In the *adjustForSlashing* function, this results in $\mathcal{P}_i$'s share being updated according to line 97, which implies $\mathbb{S}_i^{\mathcal{H}} = \frac{\mathbb{S}_i^{\mathcal{H}-1}}{1-\mathit{slashedShare}}$. Furthermore, letting $\mathit{Reward}^{\mathcal{H}-1}$ be the total reward after deciding on a value at height $\mathcal{H} - 1$, the new total reward for height $\mathcal{H}$ is $\mathit{Reward}^{\mathcal{H}} = (1 - \mathit{slashedShare})\mathit{Reward}^{\mathcal{H}-1}$ (line 98), while the total stake before distributing rewards for height $\mathcal{H}$ is adjusted to $(1 - \mathit{slashedShare})\mathit{Stake}^{\mathcal{H}-1}$ (line 99), preserving the total stake of non-deviators. We then have to add the slash bonus of $\sum_{j \in \mathit{newDeviators}^{\mathcal{H}}} \mathit{Genesis.shares}()[j]\mathit{Reward}^{\mathcal{H}}$ (line 100).

First observe that:

$$\begin{aligned}\mathbb{S}_i^{\mathcal{H}} \mathit{Reward}^{\mathcal{H}} &= \frac{\mathbb{S}_i^{\mathcal{H}-1}}{1-\mathit{slashedShare}}(1-\mathit{slashedShare})\mathit{Reward}^{\mathcal{H}-1} \\ &= \mathbb{S}_i^{\mathcal{H}-1}\mathit{Reward}^{\mathcal{H}-1}.\end{aligned} \tag{3.2}$$

Secondly, notice that when no new deviators are identified, and *adjustForSlashing* is not called, both $\mathbb{S}_i$ and Reward_i are unchanged from the previous height, as they are only adjusted in *adjustForSlashing*. This means:

$$\mathbb{S}_i^{\mathcal{H}} \mathit{Reward}^{\mathcal{H}} = \mathbb{S}_i^1 \mathit{Reward}^1 = \mathbb{S}_i^1 \mathit{Genesis.reward}(),\ \forall\ \mathcal{H} \geq 1. \tag{3.3}$$

We know for a set of new deviators $\mathit{newDeviators}^{\mathcal{H}}$ at height $\mathcal{H}$, $\mathcal{P}_i$ receives $\mathbb{S}_i^{\mathcal{H}} \mathit{Reward}^{\mathcal{H}} \sum_{j \in \mathit{newDeviators}^{\mathcal{H}}} \mathit{Genesis.shares}()[j]$(line 100). Furthermore, from Equation 3.3, we have that $\mathbb{S}_i^{\mathcal{H}} \mathit{Reward}^{\mathcal{H}} = \mathbb{S}_i^1\ \mathit{Genesis.reward}()$ for all $\mathcal{H} \geq 1$. This implies $\mathcal{P}_i$ receives an identifying deviator bonus from *adjustForSlashing* of $\mathbb{S}_i^1 \mathit{Genesis.reward}() \cdot \sum_{j \in \mathit{newDeviators}^{\mathcal{H}}} \mathit{Genesis.shares}()[j]$ at height $\mathcal{H}$. Summing over all heights up to and including $\mathcal{H}$ gives a total reward of $\mathbb{S}_i^1$ $\mathit{Genesis.reward}() \cdot \sum_{k=1}^{\mathcal{H}} \left(\sum_{j \in \mathit{newDeviators}^k} \mathit{Genesis.shares}()[j] \right)$ for identifying deviators through *adjustForSlashing*.

As $\cup_{1 \leq k \leq \mathcal{H}} \mathit{newDeviators}^k \subset \{1, ..., n\}$ for all $\mathcal{H} > 1$, it must be that $\sum_{k=1}^{\mathcal{H}} \left(\sum_{j \in \mathit{newDeviators}^k} \mathit{Genesis.shares}()[j] \right) < 1$ for all $\mathcal{H} > 1$. This means the total reward for identifying deviators in Tenderstake through the *adjustForSlashing* function is less than $\mathbb{S}_i^1 \mathit{Genesis.reward}()$, as required. □

Lemma 3.7.4. *In addition to any rewards from the adjustForSlashing function, $\mathcal{P}_i$ receives $\mathbb{S}_i^1 \mathit{Genesis.reward}()$ for every decided value in Tenderstake.*

Proof. The only reward received by $\mathcal{P}_i$ not in *adjustForSlashing* is distributed at line 69. Letting $\mathit{Reward}^{\mathcal{H}}$ be the total reward distributed at line 69 for height $\mathcal{H}$, $\mathcal{P}_i$ receives $\mathbb{S}_i^{\mathcal{H}} \mathit{Reward}^{\mathcal{H}}$. We have already seen in Equation 3.3 that $\mathbb{S}_i^{\mathcal{H}} \mathit{Reward}^{\mathcal{H}} = \mathbb{S}_i^1 \mathit{Genesis.reward}()$, for all $\mathcal{H} \geq 1$, which is the required result. □

Remark 3.7.5. *In Tenderstake, share increases are counteracted by reward decreases to keep per-decision rewards constant (Lemma 3.7.4). This avoids a common, critical, mistake in incentive compatible reward mechanisms where early share increases permanently increase the size of per-decision rewards a player receives.*

Lemma 3.7.6. *Tenderstake is SINCE in the ByRa model.*

Proof. To prove SINCE in the ByRa model, we require that every protocol action strictly dominates all other possible actions in expectation for rational players assuming all other players are rational. We do this by proving the following:

1. Rational players do not send invalid messages.

2. Rational players send valid messages when possible.

3. Rational players obey a timeout function which is increasing and unbounded in epochs at every height.

Firstly, consider invalid protocol messages. As an invalid message takes one of the forms described in Section 3.6.2, it can eventually be identified by all players and the offending player stake destroyed through the Slashing functionality. As identifying deviations of other players is strictly increasing in stake (line 100) and does not affect proceeding rewards due to Lemma 3.7.4, all rational players prefer to eventually identify valid deviations than not identify valid deviations. As stake is only meaningful with respect to a valid blockchain, $\mathcal{P}_i$ must construct $Blockchain_i$ sequentially in it's height. Therefore, for $\mathcal{H}_i$ the height of $Blockchain_i$, $\mathcal{P}_i$'s messages can only refer to a value at a height less than or equal to $\mathcal{H}_i + 1$.

Combining Lemmas 3.7.3 and 3.7.4, for any height $\mathcal{H} > 1$, the maximum additional reward achievable by sending an invalid message up to that height is less than $2\mathbb{S}_i^1\,Genesis.reward()$. This is because by Lemma 3.7.3, the additional reward for identifying deviators is less than $\mathbb{S}_i^1\,Genesis.reward()$, and by Lemma 3.7.4, the per-decided value reward excluding any reward for identifying deviators is $\mathbb{S}_i^1\,Genesis.reward()$, a constant. Therefore, attempting to decide on a value and/ or deviators (the only ways to be rewarded in Tenderstake) with an invalid message results in a payoff of less than $2\mathbb{S}_i^1\,Genesis.reward()$. Due to Lemma 3.7.4, the rewards for proceeding value-decisions remain constant, while all rewards for identifying deviators must sum up to less than $\mathbb{S}_i^1\,Genesis.reward()$ from Lemma 3.7.3. Given proofs-of-deviation can be provided at any time and all rational players will send SLASH messages when possible, the cost of sending an invalid message, full destruction of stake (line 96) and effective removal from the protocol, dominates these once-off and bounded potential rewards for sending an invalid message. As such, no rational player will send an invalid message.

Given no rational player will send an invalid message, we now need to check that rational players will send messages when valid messages can be sent, as per the protocol. The alternative is not sending messages. Given the arbitrary scheduling of message delivery in any distributed network where other players have unknown timeouts, and the positive reward for deciding on a block, sending messages strictly increases the expected rate of messages received by all other players. This in turn strictly increases the expected rate of player progression through the protocol, as progression can only occur when proofs can be generated. This strictly increases the expected number of blocks, and rewards, added to the blockchain.

Lastly, we must ensure that rational players obey a timeout function which tends to infinity in the number of epochs at each height. To do this we first show that rational players obey some non-zero timeout, and then that this timeout is increasing and unbounded in number of epochs.

If a rational player does not wait for messages to be delivered, they will never be able to contribute to prevotes for valid values unless they are a proposer. After entering a new epoch they will call line 27, immediately followed by line 76, sending a *nil* prevote. Moreover, given they send a *nil* prevote and advance to the *prevote* step, they will also send a *nil* precommit (line 80) as when they receive more than $\frac{2}{3}$ prevotes it includes their own *nil* prevote, triggering line 41 before it is possible to receive more than $\frac{2}{3}$ prevotes for a valid value. By the same argumentation, they will never be able to decide on a value in the epoch it is proposed as they will first receive more than $\frac{2}{3}$ precommits for inconsistent values given their *nil* precommit message, triggering line 57 and then immediately line 84, preventing a decision. Compare this to obeying some timeout for messages to be delivered. Waiting for some number of rounds strictly increases the probability of receiving valid proposed values, and sending a prevote for a valid value. This subsequently increases the probability of all players sending valid precommits. By further obeying a timeout for precommits it increases the probability of receiving the quorum of precommits needed to decide on a value. Therefore, rational players prefer to wait some number of rounds for messages to be delivered.

Now we must ensure rational players do not wait indefinitely for messages. Recall that in Tenderstake, rational players are modelled as assuming for some unknown but fixed Δ, they are in Δ-synchrony with some subset of players. At any round r, in order to calculate expected utility for some future round, $\mathcal{P}_i$ will have a private distribution of expected message delivery times from other players in synchrony[7], and thus an expected number of decisions up until that future round. Let τ_i^r be such that according to $\mathcal{P}_i$'s private information,

[7]The distribution of expected message delivery times will be a function of some starting estimate at initialization (perhaps based on a *Genesis* suggested value, as in [31]), and the observed responsiveness of all other players up until round r.

messages taking longer than τ_i^r are sent by players out of synchrony with $\mathcal{P}_i$ with statistical significance[8].

If the subset of players in synchrony with $\mathcal{P}_i$, including $\mathcal{P}_i$, do not control more than $\frac{2}{3}$ of the total stake, $\mathcal{P}_i$ is indifferent to timing out, as no decision is possible. Otherwise, consider the subset of players in synchrony with $\mathcal{P}_i$, including $\mathcal{P}_i$, controlling more than $\frac{2}{3}$ of the total stake. As messages sent by players out of synchrony with $\mathcal{P}_i$ take arbitrarily long to deliver, the number of decisions that can be made by using a timeout of τ_i^r and transitioning to a proposer in synchrony with $\mathcal{P}_i$ is arbitrarily large. Furthermore, as $\mathcal{P}_i$ is unaware of how many players are in synchrony with $\mathcal{P}_i$, the probability of that subset controlling more than $\frac{2}{3}$ of the stake will be positive. This implies $\mathcal{P}_i$ has positive expectancy to obey such a timeout τ_i^r. Therefore, rational players obey some timeout, and will not wait indefinitely for messages.

We finally need to show that for a rational $\mathcal{P}_i$ at any given height, $\mathcal{P}_i$ will follow increasing, unbounded timeouts in the number of epochs at every height. For a maximum message delivery time of Δ rounds during synchrony, if $\mathcal{P}_i$ follows a timeout of $\tau_i < \Delta$, $\mathcal{P}_i$ is not necessarily able to contribute to deciding on a value. Assume players controlling more than $\frac{2}{3}$ of the stake are in Δ-synchrony (if this is not the case, no information can be gained) and no decision has been made for some number of epochs. As players behave honestly in all non-timeout actions (points 1 and 2), the only variable which can affect the probability of deciding for this height is τ_i. Assume for all rational players there is a value $\tau_{max} > 0$, such that they choose timeouts less than τ_{max} for all epochs.

If $\tau_{max} < \Delta$, it is possible that players may always timeout, sending *nil* messages and not contributing to decisions. Given there has been *epoch* epochs of not deciding on a value, and all other actions are being followed (which we have shown to be the case), it must be that $P(\tau_{max} < \Delta | epoch \to \infty) \to 1$. This implies choosing a timeout up to and including τ_{max} after sufficiently many epochs of no decision results in decision with $negl(\kappa)$ probability for proceeding epochs. Therefore, rational players will eventually only follow timeouts greater than τ_{max} if no value has been decided, for any value of τ_{max}.

This is sufficient to say rational players follow increasing, unbounded timeouts, and as such, the recommended protocol. □

Lemma 3.7.7. *Tenderstake is fair in the ByRa model.*

[8]This statistical significance can be with respect to a function $negl(\kappa)$, although rational players may perceive a higher utility by choosing a weaker significance level. This optimization is unnecessary for the proof.

Proof. As all rational players follow the protocol, and $\mathbb{S}_{\mathcal{A}}^{1} < \frac{1}{3}$, no rational player decides on another rational player as deviating. Therefore, the share of stake controlled by rational players is only increasing (line 97), meaning the adversary's share is only decreasing, upperbounded by their starting share. This implies $\mathbb{S}_{\mathcal{A}}^{\mathcal{H}} \leq \mathbb{S}_{\mathcal{A}}^{1}$ for all $\mathcal{H} \geq 1$, which is precisely the definition of a fair protocol. □

Theorem 3.7.8. *Tenderstake achieves ByRa SMR.*

Proof. Follows by applying Lemma 3.7.6 and Lemma 3.7.7 to Theorem 3.5.8. □

3.8 Conclusion

We provide a game-theoretic framework for analysing SMR protocols. Although many previous attempts have been made, we are, to the best of our knowledge, the first to formally treat SMR protocols as games involving only rational and adversarial players. We detail the ByRa model for player characterization in SMR protocols, an update to the legacy BAR model, removing the dependency on altruistic players in an era of unprecedented market capitalization of tokenized SMR protocols. We demonstrate that the properties of strong incentive compatibility in expectation and fairness as described in this chapter, are both necessary, and together sufficient to achieve SMR in the ByRa model. We then provide the Tenderstake protocol as an example of a protocol that achieves ByRa SMR, which is of independent interest both as a strong incentive compatible in expectation and fair protocol in the ByRa model, but also as a yardstick for addressing the shortcomings of current protocol guarantees in the ByRa model. The proof techniques we use provide several methodologies with which SMR protocols can be analysed in this new game-theoretic framework. The improvements we make to the Tendermint protocol as described in Section 3.6 have immediate practical implications given the current industrial deployment of Tendermint-style protocols, such as in Cosmos[9].

The application of our framework to all proceeding SMR protocol analysis and development serves as critical future work. Additionally, there are several avenues for future work that arise from our choice of threat model in Section 3.6.1. Although we provide a new foundation for the game-theoretic analysis of SMR protocols, player coalitions are an important consideration from a distributed system security standpoint. Further research is required to establish

[9]Cosmos. https://cosmos.network/ Accessed: 25/05/2021

the limitations of ByRa SMR protocol guarantees in the presence of coalitions. Another important consideration is that of the ByRa model under an adaptive adversary. Investigating the effect of an adaptive adversary whose stake can increase or decrease based on protocol events makes for interesting future work.

Regarding the joining and leaving of players within the current version of Tenderstake, we envisage the necessity for stronger network communication model assumptions to ensure a player leaving the protocol has, at the very least, not sent any deviating messages that have not been resolved (and punished) within the protocol. Further research on this will be useful, especially when applying our work to more practical settings.

Chapter 4

Marvel DC

This chapter is based on the paper *Marvel DC: A Blockchain-Based Decentralized and Incentive-Compatible Distributed Computing Protocol* [76].

4.1 Introduction

The distributing of computation among computers has beckoned a never-before-seen level of computing power available to average users with access to as little as a smart phone and an average internet connection. Distributed computations (DCs) have typically been outsourced directly to one of the centralized entities such as Amazon Web Services, or Google Cloud. Unfortunately, centralized services like these have many drawbacks. Monopoly of resources, centralized trust, restricted access for many clients, and in the case of Federated Learning (FL), lack of diverse data-sets make these centralized services unfit for many users and purposes. Decentralized computation outsourcing is intended to replace the centralized computation-as-a-service model. However, existing academic solutions, whether intentionally or otherwise, typically fall back to the same trust assumptions as centralized servers; permissioned access to the computation market with dependencies on trusted parties to monitor computer outputs and to distribute rewards fairly.

Decentralizing DC protocols brings many new challenges that are largely protected against in the centralized setting. A significant issue in many existing decentralized DC implementations [71, 18, 64, 103, 101] is the accurate rewarding of computers to incentivize rational computers, computers who try to maximize their utility (e.g. in cryptocurrency tokens), to correctly perform outsourced computations. One method to combat this is, for a given computation, to use ZK tools to prove that a computer performed the actions as prescribed by the requester [97]. Although theoretically any computation can be encoded in this way, the practicality with respect to the outsourcing of generic computations in this way remains an open question.

With respect to incentivization in decentralized settings such as blockchain protocols, players only follow an action if that action is SINCE, resulting in strictly higher payoffs than the alternatives. If a computer can free-ride, submitting incorrectly computed results while receiving some positive reward in expectation, this has detrimental effects on a DC protocol. Rational computers are no longer necessarily incentivized to participate honestly, degrading the quality of computation results, which reduces requester participation which in turn reduces the overall reward pot for honest computers, eventually causing the system to collapse. Given the 10s of billions of US dollars in revenue being generated by centralized systems [8] without guarantees of fair-pricing, decentralized access or computational output quality, there is clear motivation for a decentralized DC protocol that can provide such guarantees. To this end, we present Marvel DC, which formally provides all of these guarantees, beckoning the age of decentralized DC outsourcing, free from the previously unchecked stranglehold of private monopolies.

4.1.1 Our Contribution

We present Marvel DC, a generic blockchain-based decentralized DC protocol which addresses the gaps that exist in outsourcing computations without the use of a TTP. Namely, Marvel DC provides for the first time in literature a decentralized DC protocol which ensures rational computers are strongly incentivized to follow the protocol. This is a significant advancement in a field where there remains no viable solution, to the best of our knowledge, for the distributing of tokenized rewards in a distributed and decentralized manner.

Furthermore, Marvel DC utilizes reputations to isolate correctly-performing computers when selecting computers for computations. This allows Marvel DC to efficiently remove adversarial computers from the protocol, a property formalized in Lemma 4.5.2. As these reputations are maintained on the blockchain itself, through careful construction (Section 4.4.2) this reputation protocol neither affects the decentralization or SINCE of the protocol. Our description of Marvel DC can be adapted to run on any smart-contract enabled blockchain, and as such, can make use of the vast existing communities which exist on such blockchains. This is a further improvement on protocols which require the recruitment and constant participation of an independent network of computers. In a blockchain, where computers form a subset of users, computers can be dormant until required to perform a computation. By deploying on an existing blockchain, any player in that blockchain can also participate in Marvel DC as a computer and/or requester. We summarize the main contributions of Marvel DC as follows:

SINCE: In Section 4.4, we describe how to program rewards such that it is SINCE for every rational player (requesters, computers and/or block-producers) in the blockchain system to follow the protocol. This is formalized in Theorem 4.5.1.

Handling of symmetric/asymmetric utilities: By tokenising the protocol rewards, we are able to handle both symmetric (rational computers and requesters only want to produce good results) and asymmetric (rational computers want to be compensated financially) utilities. Thus Theorem 4.5.1 can be applied to both symmetric and asymmetric utilities.

Decentralization: The description of Marvel DC in Section 4.5.1 can be translated for use on any tokenized smart-contract enabled blockchain, of which many (including Ethereum[1] and Harmony[2]) are considered truly decentralized systems due to their permissionless nature. We demonstrate this by providing a proof-of-concept implementation of Marvel DC [53] that can be deployed within such decentralized systems.

We then outline a series of privacy-enhancing amendments for Marvel DC, culminating in Privacy Marvel DC, a protocol in which results cannot be linked to the computer which provided the result, except with prohibitive additional cost to the player performing the revelation. Privacy Marvel DC retains all of the decentralization and incentive compatibility guarantees of Marvel DC, while also adding a layer of privacy which makes it appropriate for sensitive computations where knowing a certain computer computed a particular result can be used to infer private/unwanted information about the computer. This is particularly interesting in the case of FL, where computers are asked to use private data sets. To do this, we make use of existing NIZK set-membership tools. Although our privacy techniques are not novel outside of DC, the combination of decentralization, provable strong incentive compatibility and the ability to apply one smart-contract instance of Privacy Marvel DC to any computational

[1]https://ethereum.org/en/
[2]https://www.harmony.one/

problem with output in Euclidean space (summarized in Table 4.1) stands as a further novel contribution.

4.1.2 Organization of the Chapter

Section 4.2 reviews related work and presents an overview of prior attempts to implement decentralized incentive compatible DC protocols. Section 4.3 provides the background needed to outline the proceeding protocols and their properties. Section 4.4 constructs an idealized incentive compatible DC protocol, which we demonstrate in Section 4.5.1 can be implemented in a decentralized setting through the Marvel DC protocol. Section 4.5.2 formally presents this incentive compatibility via the main theorem of the chapter, Theorem 4.5.1. Lemma 4.5.2, also contained in Section 4.5.2, demonstrates that Marvel DC eventually removes misbehaving computers from consideration when selecting computers for computations. Privacy Marvel DC is then presented in Section 4.5.3. Section 4.6 provides detailed analyses of the costs and performance of these protocols, with comparisons to the state-of-the-art, as highlighted in Section 4.2, demonstrating Marvel DC and Privacy Marvel DC stand as clear improvements on existing works. We conclude in Section 4.7.

4.2 Related Work

Blockchain-based DC protocols [71, 18, 64, 103] have seen significant interest. Unfortunately, all of these papers consider a blockchain or distributed system in which all parties share one utility, that is, the ability to use/benefit from a well-trained shared model, which gives correct behaviour by definition. Such an assumption makes these protocols and their resulting analyses inappropriate in the presence of untrusted peers and/or asymmetric utilities.

In [101], a protocol and general framework for incentive mechanism design, within FL protocols where players measure utility tokenomically, are proposed. Computer registration and reward distribution must all be performed by players within the system. Computer registration is performed by an "administrator", which prevents decentralization and encourages collusion. Furthermore, computation rewards are distributed based on votes of players within the system. The incentive compatibility of this choice is not considered, and is non-trivial to implement. In Marvel DC, rewards are distributed deterministically as part of the smart-contract execution on requester inputs. We prove that rational requesters always submit messages correctly to the blockchain, and thus, correct rewarding is SINCE. Moreover, [101] has no mechanism to identify Byzantine computers and diminish/remove their ability to participate in the protocol. In Marvel DC, this is achieved using reputations.

Protocol	Tokenized Rewards	SINCE	Computation Independent	Diminishing Adversarial Selection Prob.
[71, 18, 64, 103]	✗	✗	✓	✓/ ✗
Toyoda et al. [101]	✓	✗	✓	✗
Ruckel et al. [97]	✓	✓	✗*	✗
Marvel DC	✓	✓	✓	✓

Table 4.1: Comparison of incentive-aware distributed computing protocol designs. *Computation independence refers to the ability of a particular smart-contract encoding to be re-used for many computations. The ZK circuit-encodings of [97] must be generated anew for each type of computation, placing significant upfront costs on computation requesters.

Recently, an extensive survey of existing attempts to construct privacy-preserving FL protocols[72] was published. Of the investigated works, the most promising for achieving an incentive-compatible decentralized protocol is [97]. In [97], ZK-proofs are used in the computing stage to prove that a computation was performed correctly. In a decentralized setting however, this raises many challenges, as each type of computation requires its own ZK circuit-generation/trusted set-up to generate the target function. In contrast, Marvel DC can be implemented using existing, blockchain-deployed ZK-tools and generalised rewarding functions. Moreover, as acknowledged by the authors of [97], although their methodology ensures models were trained correctly, it does not guarantee the models were trained on appropriate data. A proposed solution is using "certified sensors", equivalent to TTPs, a non-viable solution in a decentralized setting. As the rewarding functions in Marvel DC reward players based on the relative quality of the results, and not just based on the fact that a series of computations has been performed correctly, independent of the data on which the computations are being performed, we are able to avoid this issue.

4.3 Preliminaries

This section introduces additional terms and notation used to describe Marvel DC. We are interested in a distributed set of n players $\{\mathcal{P}_1, ..., \mathcal{P}_n\}$ interacting with one and other inside a blockchain protocol. These players send and receive stake among one another, along with the functionality to encode programs to run within the blockchain protocol in the form of *smart contracts*. In this chapter, we assume the ByRa player model as defined in Definition 3.4.1. The ByRa model is a necessary improvement on the legacy BAR model [70] for the true consideration of incentives in distributed systems, removing any dependencies on altruistic, honest-by-default players which cannot be assumed to exist in incentive-driven protocols like blockchain/DC protocols.

It is the aim of this chapter to construct a DC protocol that ensures rational players always follow the protocol, a property known as strong incentive compatibility in expectation. In this chapter, rational player utility is measured in blockchain-based tokens. The blockchain protocol acts conceptually as a public ledger managed by a TTP. In reality, it is the following of the blockchain protocol by some majority of players using the blockchain that replicates this TTP. The blockchain protocol provides availability and correctness of the programs being run by the blockchain protocol, but does not provide privacy. That is, any player can observe the current state of all programs being run on the blockchain, and can verify that this state has been reached through the correct running of these programs. However, player inputs to these programs must be committed publicly to the blockchain before they can be passed to the smart contract, and as such, it will be an important requirement of designing a protocol involving smart contract interaction, through transactions, that the blockchain will accept these transactions in a timely fashion. In our system, this is achieved using incentivization.

4.4 Constructing a SINCE DC Protocol

We first describe an idealized protocol for the distribution of computation between a set of computers. We then demonstrate how to instantiate such a protocol using existing blockchain technology. In this description, we consider a requester who has a computation *calc* whose calculation the requester seeks to outsource to some subset of available computers $\mathfrak{C}$. Furthermore, the requester is aware of a threshold n_ψ such that for any random sample of n_ψ computers without replacement from $\mathfrak{C}$, a majority of those computers are rational (see Table 4.2).

We consider the output of all computations in this chapter as a point in l-dimensional space. To reason about the goodness of a computation result, for every computation we assume the existence of a deterministic function which takes the set of computation responses, and given a majority of correctly computed results, outputs a target value $\varkappa$, which is computed correctly with probability $1 - \epsilon$, for some $\epsilon < 0.5$. For deterministic calculations, the mode is such a function. In FL where results are model gradients, the Krum function [29] is such an aggregation function.

Consider the set of computation results $\{result_1, ..., result_{n_{comp}}\}$. We require for any pair $(result_+,\ result_-)$, with $result_+$ a correctly computed result and $result_-$ an incorrectly computed result the following holds:

$$P(distance(result_+, \varkappa) < distance(result_-, \varkappa)) > 0.5. \tag{4.1}$$

Given this, for any subset of computation results taken in ascending order using the function *distance*, the expected number of these computations being correctly computed is greater than those being incorrectly computed. With respect to deterministic computations, letting $\varkappa$ be the median or mode, all correctly computed results will be distance 0 to $\varkappa$. For non-deterministic computations, it is less clear. The function used to compute $\varkappa$ must be chosen such that Equation 4.1 holds.

Now, consider the following DC protocol run by a TTP who enforces the correct participation of all rational players. The proceeding sections then replace this TTP, in order to achieve full decentralization, through the use of a smart contract-enabled blockchain and a strong incentive compatible protocol (Marvel DC, described in Section 4.5) to be run therein.

Idealized distributed computing protocol. Requesters repeatedly enter the system with independent functions for computation. A requester wishing to avail of the distributed computation of some function *calc* submits *calc* to the TTP, as well as some number of computers $n_{comp} > n_{\psi}$. The TTP then selects n_{comp} from the set of available computers $\mathfrak{C}$. The computers respond to the TTP with their computation of *calc*, who then sends all of the computation results to the requester.

In a distributed setting, such TTPs do not exist. Therefore, to enforce the correct participation of rational players we need to utilize a mixture of cryptography and incentivization. With this in mind, we first describe how to generically construct a reward mechanism such that rational players are strongly incentivized to follow the protocol. We then outline a reputation management protocol which maintains this incentivization, while reducing the probability that incorrectly performing computers are selected for computation.

4.4.1 Reward Mechanism

We now provide a theoretical lower-bound on the per-computer reward required to strongly incentivize the correct participation of rational computers in our DC protocol. Running computations such as sorting arrays or encryption/decryption on-chain is expensive, so we initially give the requester the opportunity to correctly submit the set of computers to reward. Computers have an opportunity to contest this by depositing a bounty on-chain, triggering the on-chain verification of the reward set. Note that verification is much cheaper than computation, but with respect to Privacy Marvel DC, this reveals some private information, which is why verification does not take place automatically. If the contest is valid, the requester loses some initial escrow, herein lower-bounded. Otherwise, the computer loses the bounty. This is described in more detail in Section 4.5.

For a given computation *calc*, we assume an accurate a-priori lower-bound on the cost to compute a particular computation *calc* of $\mathit{cost}(\mathit{calc})$. This lower-bound is known by all players in the system (in reality this value can be enforced by the protocol smart-contract). Given that the payment of $\mathit{Fee}(\mathit{tx})$ per transaction guarantees timely inclusion in the blockchain, rational computers perform the calculation of *calc* if and only if:

$$E(\mathit{Reward}(\mathit{calc})) > \mathit{cost}(\mathit{calc}) + 2\mathit{Fee}(\mathit{tx}). \tag{4.2}$$

$2\mathit{Fee}(\mathit{tx})$ is needed as computers are required to send 2 messages to the blockchain.To more accurately describe $\mathit{Reward}(\mathit{calc})$, we introduce $0 \leq \omega, \gamma \leq 1$ with the probability of good computations being rewarded being ω, and the probability of bad computations being rewarded being γ, such that without loss of generality $\omega > \gamma$. This gives an expected payoff of $\omega \mathit{Reward}(\mathit{calc}) - (\mathit{cost}(\mathit{calc}) + 2\mathit{Fee}(\mathit{tx})) > 0$ for following the protocol, and an expected payoff of $\gamma \mathit{Reward}(\mathit{calc}) - 2\mathit{Fee}(\mathit{tx})$ for submitting a result but not performing the computation. Therefore, we require:

$$\omega \mathit{Reward}(\mathit{calc}) - \mathit{cost}(\mathit{calc}) - \gamma \mathit{Reward}(\mathit{calc}) > 0 \tag{4.3}$$

This reduces to $\mathit{Reward}(\mathit{calc}) > \frac{\mathit{cost}(\mathit{calc})}{\omega - \gamma}$. The exact values of ω and γ depend on the computation, number of computers to be rewarded and the chosen target function. Exact values for ω and γ are difficult to predict a-priori. For deterministic computations $\omega \approx 1$, whereas for non-deterministic computations such as FL, ω will be smaller. Lower-bounding the possible value of $\omega - \gamma$ (although greater than 0) and using this value to compute $\mathit{Reward}(\mathit{calc})$ ensures rational players follow the protocol.

To ensure a rational requester correctly submits the set of good computations to be rewarded, any positive escrow amount $escrow_{req} > Fee(tx)$ suffices to strongly incentivize the requester to do this. This can be seen by considering the payoff for submitting the correct set, which is $escrow_{req} - Fee(tx) > 0$, while the payoff for not submitting the set is 0. In the case of potential collusion of up to k computers, setting $escrow_{req} \geq k \cdot Reward(calc) + Fee(tx)$ guarantees that the requester correctly submits the set of good computations. If k is set too small by the protocol/smart contract, not submitting the reward set, and rewarding all players may be positive expectancy for the requester. Setting k equal to n_{comp} conservatively achieves this. With this lower bound on $escrow_{req}$, rational requesters always submit the correct set of good computations to the blockchain.

4.4.2 Reputation Management Protocol

In this section we describe a reputation management protocol that maintains the incentive compatibility of a DC protocol with an incentive compatible reward mechanism, while also allowing us to prioritize good computers over bad computers, meaning both short- and long-term benefits for correctly behaving computers. We consider a rating mechanism $rate()$ which, under the same assumptions of the previous section, assigns good calculations a score of 1, and bad calculations a score of 0. For a player $\mathcal{P}_i$ taking part in computations for $calc_1, calc_2, ..., calc_k$, $\mathcal{P}_i$'s *base reputation* is $baseRep_i = \sum_{j=1}^{k} rate(calc_j)$.

Let $initalRep > 0$ be the starting reputation for computers registering in the system. For a given computation, we let $\mathcal{P}_i$ be selected as computer for a computation in block at height $\mathcal{H}$ in the blockchain in direct proportion to $baseRep_i^{\mathcal{H}-1}(initalRep - 1)$ as a fraction of $\sum_{j=1}^{n} baseRep_j^{\mathcal{H}-1} - n \cdot (initalRep - 1)$. With this in mind, the number of computations a player $\mathcal{P}_i$ is selected for is directly proportional to:

$$probSelect_i^{\mathcal{H}} = \frac{baseRep_i^{\mathcal{H}-1} - (initalRep - 1)}{\sum_{j=1}^{n} baseRep_j^{\mathcal{H}-1} - n \cdot (initalRep - 1)}. \tag{4.4}$$

Consider a player $\mathcal{P}_i$ who includes a good computation as block proposer for a computer $\mathcal{P}_j$. This increases $\mathcal{P}_j$'s base reputation, and thus $\mathcal{P}_j$'s *probSelect*, which has long-term stake implications (more selection = more reward & more reputation = more selection ...). This negatively affects the *probSelect* of $\mathcal{P}_i$ however. Therefore, we need to reward the proposer with an increase in base reputation to counteract the increase in the computers' expected increase in base reputation. Let $E_{rep} > 0$ be the expected change in base reputation for a computer whose computation gets included on the blockchain.

For $\mathcal{P}_i$ a proposer of a block that includes k transactions containing computation results, we need the following equality to hold:

$$\begin{aligned} probSelect_i^{\mathcal{H}} &= \frac{baseRep_i^{\mathcal{H}} - (initalRep - 1)}{baseRep_i^{\mathcal{H}} + \sum_{j \neq i} baseRep_j^{\mathcal{H}-1} + k \cdot E_{rep} - n \cdot (initalRep - 1)} \\ &= \frac{baseRep_i^{\mathcal{H}-1} - (initalRep - 1)}{baseRep_i^{\mathcal{H}-1} + \sum_{j \neq i} baseRep_j^{\mathcal{H}-1} - n \cdot (initalRep - 1)}. \end{aligned} \tag{4.5}$$

Solving for $baseRep_i^{\mathcal{H}}$ in this equality gives:

$$baseRep_i^{\mathcal{H}} = baseRep_i^{\mathcal{H}-1} + k \cdot E_{rep} \Big(\frac{baseRep_i^{\mathcal{H}-1} - (initalRep - 1)}{\sum_{j \neq i} baseRep_j^{\mathcal{H}-1} - (n-1) \cdot (initalRep - 1)} \Big). \tag{4.6}$$

This means we need to add $k \cdot E_{rep} \big(\frac{baseRep_i^{\mathcal{H}-1} - (initalRep-1)}{\sum_{j \neq 1} baseRep_j^{\mathcal{H}-1} - (n-1) \cdot (initalRep-1)} \big)$ to $baseRep_i^{\mathcal{H}-1}$ in order to ensure the proposer is impartial, with respect to reputation and computer selection probability, to adding transactions containing computation results to the blockchain. For transactions from the requester finalising the rewards, we simply have to replace E_{rep} with the actual mean change in reputation in Equation 4.6, and the rest of the numbers stay the same.

4.5 Marvel DC

The goal of this section is to take the ideal DC functionality of Section 4.4 and it's properties, and implement them as a set of algorithms encoded as smart contracts that can be run by a decentralized set of players without a TTP, but with access to a blockchain. We call this protocol Marvel DC. We then formally prove in Section 4.5.2 that within Marvel DC, rational computers always follow the protocol, performing computations correctly, through strong incentive compatibility. As demonstrated in Section 4.2, this guarantee stands alone in the field of decentralized DC. Furthermore, as Marvel DC is provided in a generic manner, it can be deployed on any decentralized blockchain, allowing the Internet of Things never-before-witnessed access to DC.

4.5.1 Algorithmic Overview

Marvel DC is a set of smart contracts provided in pseudocode in Algorithms 2, 3 and 4. The main contracts corresponding to unique phases in the protocol are labelled *Register*, *Request*, *Response* and *Finalize*. A proof-of-concept Solidity implementation of Marvel DC has been made available on Github [53]. We provide here the intuition to these encodings.

A Marvel DC instance is initialized by calling the Request contract, and lasting at least T blocks, where T is the number of blocks required for players to observe an event on-chain, send a transaction and have that transaction committed on-chain given at least $Fee(tx)$ is paid. We provide here the intuition to these encodings. Each player $\mathcal{P}_i$ has exclusive access to a token balance bal_i which is stored as a globally readable variable on the blockchain. For a token ₿, bal_i(₿) is the amount of token ₿ that $\mathcal{P}_i$ owns. Players in the underlying blockchain protocol can enter Marvel DC as computers by calling the Register contract, which for a given computer deposits an escrow $escrow_{comp}$ (line 44), granting that computer a reputation of *initalRep* (line 46).

A computation request specifies the computation details *calc*, the number of computers to be selected for the computation n_{comp}, a deterministic function $f_{\varkappa}$ for selecting the target result $\varkappa$ from the set of results, the number of computers to reward n_{reward}, and the per-computer reward $Reward_i$ received by a computer if included in the set of computers to reward. The requester also must deposit an escrow of $n_{reward} \cdot Reward_i + escrow_{req}$. The *compEncKey* is the public key corresponding to the temporary public/private key pair (*compEncKey*, *compDecKey*). This is a key pair generated by the requester specifically for *calc*. A randomness beacon is called (line 50), which provides a pseudo-random seed for selecting n_{comp} computers to participate in the computation in direct proportion to computer reputations (line 5), with these computers listed in the set $calc.\mathfrak{C}$.

Selected computers can then submit results for *calc* to the blockchain by calling the Response contract, with results encrypted using *calc.compEncKey*. This encryption ensures no other computer can use another computer's result, and therefore must themselves perform the computation. Given a valid response is recorded, the block producer corresponding to the response is added to *calc.proposers*. This is later used to update reputations, in line with the analysis of Section 4.4.2.

Then, either after T blocks from when the computation *calc* was requested, or when all computers in $calc.\mathfrak{C}$ have responded, the requester of *calc* can complete the request by calling the Finalize contract. Calling the Finalize contract requires the requester to provide the decryption key *calc.compDecKey* corresponding to *calc.compEncKey.* If these keys correspond to a valid key pair, the requester receives back her escrow $escrow_{req}$. The contract then uses *compDecKey* to decrypt the computer responses, and identify which computers are to be rewarded (line 19). This is done by applying the pre-specified target function to the computation results, and computing a target value $\varkappa$ (line 22). The computers corresponding to the $calc.n_{reward}$ results closest to $\varkappa$ (using Euclidean distance in the provided pseudocode) are selected as the computers to reward, $\mathfrak{C}_{good}$ (line 23). The computers in $calc.\mathfrak{C}_{good}$ each receive $calc.Reward_i$. Finally, all proposers in *calc.proposers* who are also registered computers receive reputation increases of *avgRepChange* (line 30), while computers in $calc.\mathfrak{C}_{good}$ each receive an increase in reputation of 1 (line 34).

4.5.2 Protocol Properties

This section takes the Marvel DC protocol described in Section 4.5.1, and demonstrates its SINCE (Theorem 4.5.1), and the long-term benefit provided by the reputation protocol (Lemma 4.5.2) used therein: namely, that Byzantine computers not performing computations correctly are selected with diminishing probability.

In the following, we consider rational requesters idiosyncratically entering the system, running unique instances of the Request contract. Given this, we first show that rational computers and rational requesters are strongly incentivized to participate in the protocol.

Theorem 4.5.1. *There is a strict Nash Equilibrium in which, for any computation with a per player reward* $Reward_i > \frac{cost(calc)}{\omega-\gamma}$, *rational computers and requesters follow the Marvel DC protocol.*

Proof. Consider a Request(*requester*, $*$) instance corresponding to a computation *calc*, and computers selected for computation $calc.\mathfrak{C}$. Based on $n_{comp} > n_{\psi}$, the majority of computers in $calc.\mathfrak{C}$ are rational.

First consider a rational requester. Correctly running Finalize (*calc*, $*$) allows the requester to receive back $calc.escrow_{req}$, and as such, rational requesters follow the protocol. Consider now rational computers. If the requester correctly runs Finalize (*calc*, $*$), then $calc.\varkappa$ and $calc.\mathfrak{C}_{good}$ are generated correctly. Therefore, if all rational computers follow the protocol, the assumption under which we chose $Reward_i$ in Section 4.4, for a given rational computer $\mathcal{C}_i$ correctly running Response(*calc*, $*$), $\mathcal{C}_i$ is included in $calc.\mathfrak{C}_{good}$ with probability ω. If $\mathcal{C}_i$ incorrectly runs Response(*calc*, $*$), $\mathcal{C}_i$ is included in $calc.\mathfrak{C}_{good}$ with probability of at most γ. By our choice of $Reward_i$, we have seen in Section 4.4, given $calc.\mathfrak{C}_{good}$ is generated correctly and computers included in $calc.\mathfrak{C}_{good}$ receive this with probability 1, this is sufficient for rational computers to compute the result correctly, equivalent to calling Response($calc, *$). Therefore, rational computers and requesters follow the protocol if $\mathit{Reward}_i > \frac{cost(calc)}{\omega - \gamma}$. □

This result is enough to ensure rational players follow the Marvel DC protocol. As such, a majority of results are correctly computed for every computation, agnostic to the semantics of the results. Requesters are still required to perform due dilligence when using these results as some minority may have been produced maliciously.

A consequence of using the reputation and computer-selection mechanism as described in Section 4.4.2, Marvel DC also guarantees that Byzantine computers are selected with diminishing probability in the number of computations, converging to 0 for any minority of selected computers. This is stated formally in the following lemma.

Lemma 4.5.2. *For a series of computations $[calc_1, calc_2, ..., calc_i]$ in Marvel DC with $\mathit{Reward}_i > \frac{cost(calc)}{\omega - \gamma}$ and $n_{comp} > n_{\psi}$, as the number of completed computations increases, the probability of selecting a Byzantine computer for a computation with $n_{comp} < \frac{|\mathfrak{C}|}{2}$ is strictly decreasing in expectancy and converges towards 0 as i tends to infinity.*

Proof. As $\mathit{Reward}_i > \frac{cost(calc)}{\omega - \gamma}$, from Theorem 4.5.1 rational computers follow the protocol. Let α be the share of computers that are Byzantine. We know a majority of computers selected are rational, as $n_{comp} > n_{\psi}$. Therefore, Byzantine computers are rewarded with probability $\gamma < \omega$. For a given computation, the expected reputation increase of a selected Byzantine computer is γ, while the expected increase for a selected rational computer is ω. Given n_{comp} are selected for the computation, the expected number of these being rational computers is $(1 - \alpha)n_{comp}$, while the number of selected Byzantine computers is αn_{comp}. Furthermore, this means the expected increase in reputation for rational computers is $(1 - \alpha)n_{comp}\omega$, while the expected increase in reputation for Byzantine computers is $\alpha n_{comp}\gamma$. At the beginning of the protocol, the probability of selecting a Byzantine player from the set of all computers is in direct proportion to starting reputation. Given initial reputations of *initalRep*, after the first computation, the selection probability of a Byzantine computer reduces in expectancy to:

$$\frac{\alpha(|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\big((1 - \alpha)\omega + \alpha\gamma\big)}. \tag{4.7}$$

First it be can see that

$$\frac{\alpha(|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\big((1-\alpha)\omega + \alpha\gamma\big)} < \alpha \tag{4.8}$$

meaning Byzantine selection probability is decreasing. To prove that Byzantine selection probability tends to 0 in the number of computations as described in the Lemma statements, let α_k be the Byzantine computer selection probability after k computations. We have the expected Byzantine selection probability after $k+1$ computations, denoted , α_{k+1}, is:

$$\begin{aligned} &\frac{\alpha_k(|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\big((1-\alpha_k)\omega + \alpha_k\gamma\big)} \\ =& \frac{\alpha_k(|\mathfrak{C}| \cdot \mathit{initalRep} + \gamma n_{comp})}{|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\omega - \alpha_k n_{comp}(\omega-\gamma)}. \end{aligned} \tag{4.9}$$

We have already seen α_{k+1} equals

$$\frac{\alpha_k(|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\omega - \alpha_k n_{comp}(\omega-\gamma)} < \alpha_k. \tag{4.10}$$

which implies:

$$\frac{(|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\gamma)}{|\mathfrak{C}| \cdot \mathit{initalRep} + n_{comp}\omega - \alpha_k n_{comp}(\omega-\gamma)} < 1. \tag{4.11}$$

Letting the term on the left be r_k, we can see r_k is decreasing in k as:

- $n_{comp}(\omega-\gamma) > 0$ (because $\omega > \gamma$).
- $0 < \alpha_{k+1} < \alpha_k$.

These together mean the negative term in the denominator of r_k, $\alpha_k n_{comp}(\omega - \gamma)$, is increasing (towards 0) and as such the denominator of r_k is increasing. Therefore $\alpha_k < \alpha_0 r_0^k$, with $r_0 < 1$. The result follows. □

Lemma 4.5.2 depends on the output of an on-chain randomness oracle being unpredictable when the Request contract is called. A requester who knows the input seed *randomSeed* to the Marvel DC computer selection protocol (line 5) can select Byzantine computers disproportionately, and use this to articially increase the reputations of Byzantine computers. With a random input seed, Marvel DC randomly draws from the set of computers in direct proportion to reputations. Existing randomness solutions, such as the Chainlink Verifiable

Random Function [3], provide proofs that a provided randomness was generated correctly. Given an oracle that can produce randomness periodically, not necessarily related to Marvel DC, the guarantees of Lemma 4.5.2 hold. Analysis of the quality of on-chain randomness is beyond the scope of this thesis.

As a direct consequence of Lemma 4.5.2, with reasonable choices for rewarding functions and number of computers per-computation (explored in Table 4.2), both enforceable by the protocol, Byzantine players are eventually removed from the system. This improves the efficiency of the protocol over time, reducing the minimum requirements for computers, and as such, latency, transaction fees, and rewards.

	n_{comp} =**10**	**25**	**100**	**1000**
α =**0.45**	4.9×10^{-1}	3.1×10^{-1}	1.8×10^{-1}	8×10^{-4}
0.33	2.1×10^{-1}	4.2×10^{-2}	4.2×10^{-4}	0
0.2	3.3×10^{-2}	3.7×10^{-4}	2.1×10^{-11}	0
0.1	1.6×10^{-3}	1.6×10^{-7}	0	0
0.05	6.4×10^{-5}	3.6×10^{-11}	0	0
0.01	2.4×10^{-8}	0	0	0

Table 4.2: Approximate probability of not choosing a majority of rational computers given specific starting adversarial % of computers α (left column) and selected numbers of computers n_{comp} (top row). These probabilities assume a sufficiently large population of computers such that adversarial share represents the per-selection probability of selecting an adversarial computer throughout sampling. We let 0 represent any positive number less than 10^{-14} due to precision constraints.

.

4.5.3 Privacy Marvel DC

In this section we outline a privacy enhancement to Marvel DC which we call Privacy Marvel DC. The motivation for this enhancement is to allow for an additional level of computer privacy which can be seen as necessary in computations such as those in FL protocols. This additional privacy on top of the novel contributions of being SINCE, generically applicable and fully decentralized further add to the applicability and utility of our work in an even larger set of DC problems.

[3] https://docs.chain.link/docs/chainlink-vrf/

We present Privacy Marvel DC by describing it's key differences to Marvel DC to ensure that in an optimistic scenario, only the requester and computers involved in a computation learn the results, and that players in the system can at most infer a computer submitted a good result (or bad result), but not which of the good results (bad results). In the pessimistic scenario, all players in the blockchain observe the results, but it still holds that any player in the system can at most infer a computer submitted a good result (or bad result), but not which of the good results (bad results). In Privacy Marvel DC, there is an additional contract, *Reveal*, which is to be executed after the Response contract, and before rewards are finalized. The purpose of the Reveal contract is described later in this section.

During computer registration, computers in Privacy Marvel DC privately generate $\mathcal{S}_1,\ \mathcal{R}_1 \in \{0,1\}^{\Theta(\kappa)}$, and attach $regID_1 \leftarrow commit(\mathcal{S}_1, \mathcal{R}_1)$ to the registration message, as described in Section 2.3, information necessary to prove set membership at some later point. Then, when a requester requests a computation, and the indices for computation $calc.\mathfrak{C}$ are calculated, the requester now generates a set containing the indices as specified in $calc.\mathfrak{C}$, a set to which only computers in $calc.\mathfrak{C}$ can prove NIZK set membership. In Privacy Marvel DC, this allows for the separation of result submission and player identity.

In addition to the deposits of Marvel DC, the requester must also deposit a pool of money necessary to incentivize relayers (Section 2.4) to publish transactions on behalf of computers involved in the computation. Given the amount of money required by one relayer to include a blockchain transaction is fee_r, the additional required deposit is $n_{comp} \cdot fee_r$ for the relaying of computer messages during the Response phase.

In the Response contract, a computer selected in $calc.\mathfrak{C}$ can submit a NIZK proof of membership in $calc.\mathfrak{C}$. Such a computer must also generate and submit a new $\mathcal{S}_2,\ \mathcal{R}_2 \in \{0,1\}^{\Theta(\kappa)}$ pair, and compute $regID_2 \leftarrow commit(\mathcal{S}_2, \mathcal{R}_2)$. Setting $m \leftarrow \langle calc, response, \mathcal{R}_1, regID_2 \rangle$, the computer generates a NIZKSoK $\pi_1 \leftarrow NIZKSoK[m]\{(regID_1,\ \mathcal{R}_1) : \text{MemVerify}\ (calc.\mathfrak{C},\ regID_1) = 1\ \ \&\ \ regID_1 = commit(\ \mathcal{S}_1,\ \mathcal{R}_1)\ \}$. Finally, the computer then publishes m and π_1 to the blockchain through a relayer, who receives fee_r upon the transactions addition to the blockchain.

In the Reveal contract, the requester off-chain performs the same calculations that were done on-chain in Marvel DC to calculate the results to be rewarded. Instead of adding computer indices to $responses_{good}$, the requester adds the corresponding $regID_2$s. The requester publishes $responses_{good}$ and the encryption of $calc.compDecKey$ using each public key corresponding to computers in $calc.\mathfrak{C}$ to the blockchain. However, rewards are not immediately distributed to computers in $responses_{good}$.

In the Finalize contract, computers can choose to contest the computation of $responses_{good}$ for up to T blocks after $responses_{good}$ was published. If $responses_{good}$ was computed incorrectly, any of the computers in $calc.\mathfrak{C}$ can publish the decryption of all results, and in-so-doing prove $responses_{good}$ was incorrectly computed of by the requester. In this case, all computers are rewarded, and the requesters escrow is destroyed. To prevent malicious computers in $calc.\mathfrak{C}$ from attempting this, a further escrow is required, which is returned on the correct proving of miscomputation of $responses_{good}$ by the requester.

If $responses_{good}$ was computed correctly, any computer whose $regID_2$ is included in $responses_{good}$ can generate a proof of membership to $responses_{good}$. Furthermore, as $regID_1$ can no longer be used for future computations (using the same $regID_1$ would reveal the same $\mathcal{R}_1$ in the next $calc$), computers must now generate a new $\mathcal{S}_3, \ \mathcal{R}_3 \in \{0,1\}^{\Theta(\kappa)}$ pair and corresponding $regID_3 \leftarrow commit(\mathcal{S}_3, \mathcal{R}_3)$.

Observation

As computers submit results through a relayer, and with an accompanying NIZKSoK π proving membership in the selected indices for computation, all players in the blockchain protocol can be sure the player submitting the message must be a selected computer. Crucially, nothing else can be learned about the submitting player's identity. Similarly, when collecting rewards, or replacing $regID_1$, the only thing learned when a computer submits a valid message during the Finalize phase is to which set, $responses_{good}$ or $responses \backslash responses_{good}$, the computer belongs.

4.5.4 Further Privacy Enhancements

There are further privacy enhancements possible for Marvel DC. One such enhancement is to detach reward collection/reputation updates from the computation. Given $responses_{good}$ was calculated correctly, computers included in $responses_{good}$ can instead delay their claiming of the reward and associated reputation increase arbitrarily. After a Finalize phase without arbitration, the set of $regID_2$s corresponding to $responses_{good}$ can be added to a pool of all recorded good responses throughout the protocol. These can then be immediately/periodically/sporadically claimed by computers, depending on the privacy requirements of the computer in question. This again uses the same NIZK set-membership techniques, except now with a larger set in which to diffuse.

4.6 Implementation Analysis

In this section we analyse the costs and performance of Marvel DC and Privacy Marvel DC. We show that, on top of the unique decentralized incentive compatibility guarantees of Section 4.5.2, both protocols are cost-effective and practical for computers and requesters. The proof-of-concept encodings of Marvel DC used for this analysis are available here [53].

4.6.1 Gas cost of running Marvel DC and Privacy Marvel DC

There are several considerations when calculating the cost of running Marvel DC on a blockchain. In the case where a computation has a single 256-bit answer, the costs are significantly less than in the case of gradient estimation problem where answers contain thousands or millions of numbers. More concerning still is the prohibitive nature of messages in the order of MBs in many blockchain protocols. To counteract this, messages for the Response and Finalize contracts can be submitted to memory-efficient alternatives such as IPFS[4], Layer-2 chains or even through an MPC protocol between computers and requesters.

In Table 4.3 we present, for various values of n_{comp} and n_{reward}, the approximate gas and US dollar costs (using the Harmony blockchain[5]) for Marvel DC and Privacy Marvel DC given a 256-bit result, with all messages published on-chain. This can be extended to l-dimensional results for any $l > 1$. The key takeaway from Table 4.3 is that the running costs for players to decentralize their computational resources and requests in (Privacy) Marvel DC are practically negligible, being less than \$0.01 for any individual in the examples provided.

Thus, the primary consideration for players in (Privacy) Marvel DC is accurately estimating the costs for computation. This can be done through a price-discovery process between protocol participants (players vote/bid on the cost of performing particular computations), potentially involving on-chain energy price oracles[6] to estimate the cost of particular types of computations without active participation from protocol participants.

The set-membership tools described in Privacy Marvel DC are pre-compiled, and currently being used in the Tornado Cash privacy protocol. We thus cal-

[4]https://ipfs.io/

[5]https://explorer.harmony.one/, Accessed: 17/08/2022

[6]Crude Oil price oracle https://data.chain.link/ethereum/mainnet/commodities/wti-usd, Accessed: 18/08/2022

culate the gas cost of the set-membership tools using the Tornado Cash library. All other operations are typical on-chain array manipulation, encryption/decryption, deposit, withdraw and writing to memory operations. We also include the cost of calling an on-chain randomness oracle in our calculation. This call needs to be made prior to the calculation of the indices for computation, and must be made when depositing rewards and escrow to ensure the requester cannot manipulate the selection of computers.

4.6.2 Performance metrics

A direct comparison of our protocol to existing distributed FL solutions under the headings of latency, throughput and communication complexity is of limited benefit. The primary reason for this is our protocol does not require protocol players to manage the blockchain, while previous standalone solutions [71, 18, 64, 103] must ensure that a majority of nodes involved in the protocols are efficiently communicating. It is still possible in decentralized blockchain protocols to participate in the blockchain consensus protocol, although monitoring services and the payment of gas fees allow individual participants to avoid this. In a decentralized blockchain setting, protocol participants are only required to listen for messages and events relevant to themselves. As such, decentralized protocols with similar scope to our work [101, 97] avoid such comparisons, as do we.

	Marvel DC		Privacy Marvel DC		
$\{n_{comp}, n_{reward}\}$	Comp.	Req.	Comp.-No Arbitration	Comp.-Arb.	Req.
$\{5, 1\}$	60 (0.19)	763 (2)	828 (3)	541 (2)	1,429 (5)
$\{5, 3\}$	60 (0.19)	777 (2)	828 (3)	541 (2)	1,569 (5)
$\{10, 2\}$	60 (0.19)	918 (3)	828 (3)	541 (2)	2,686 (9)
$\{10, 5\}$	60 (0.19)	960 (3)	828 (3)	541 (2)	3,176 (10)
$\{25, 5\}$	60 (0.19)	1,425 (5)	828 (3)	541 (2)	7,868 (25)
$\{25, 13\}$	60 (0.19)	1,537 (5)	828 (3)	541 (2)	9,729 (31)

Table 4.3: Amortized gas costs in 1,000s for computers and requesters in Marvel DC and Privacy Marvel DC for several choices of computers to select n_{comp} and computers to reward n_{reward}. The equivalent costs of using the Harmony blockchain in thousandths of a US dollar are included in round brackets.

However there are comparisons with [101, 97] that we can perform. Using the terminology of this thesis, every protocol *phase*, a period of time where an event occurs which requires a response, lasts up to T blocks. These T blocks (as used in Section 4.5.1) are equivalent to the time required to ensure players can submit transactions to the blockchain after a particular on-chain event, such as a computation request. Marvel DC therefore lasts up to $2T$ blocks, which covers the time for computers to respond to a request, and the time taken for the requester to reveal the decryption key. The protocol of [101] lasts at least $4T$ blocks to ensure workers are incentivized to submit at least one correct model update (using the terminology of [101], 2 model update rounds are needed for this to be the case). The additional costs are due to requesters and computers being required to respond twice each after the initial request. The protocol of [97] requires at least $3T$ blocks, as computers must commit to the data-set to be used, before submitting a computation result. All 3 protocols, including our own, are equipped to spawn arbitrarily many computations in parallel.

4.7 Conclusion

We present Marvel DC, a SINCE blockchain-based decentralized DC protocol which stands as a new standard in constructing fully decentralized DC protocols. This is achieved through a novel combination of strong incentivization of rational computers in the presence of Byzantine computers and reputation-aware computer selection. Furthermore, we outline Privacy Marvel DC, which utilizes privacy-enhancing techniques that can be bootstrapped to the core Marvel DC protocol to allow for computations on sensitive data without compromising the privacy of the computers participating in the protocol. We demonstrate in Section 4.6 that, in addition to the unique incentivization guarantees of Marvel DC and Privacy Marvel DC exhibited in Section 4.5, these protocols are cost-effective and ready to deploy, while providing provable performance and running-time improvements on existing state-of-the art.

Much work remains to ensure DC protocols remain incentive compatible and practical where computations produce large outputs, with storage being a limiting resources in mainstream blockchain protocols. Marvel DC and Privacy Marvel DC serve as key protocols with which to continue this research.

Algorithm 2 Computer Selection Protocol

```
1: function $genProbSelect()$
2:     $minRep \leftarrow min(Reps)$
3:     $denominator \leftarrow \sum(Reps) - length(Reps) \cdot (minRep - 1)$
4:     return $[((i - (minRep - 1))/denominator)\ for\ i \in Reps]$    ▷ Probability formula from Section 4.4.2

5: function $selectComputers(randomSeed, n_{comp})$
6:     $ctr \leftarrow 0$
7:     $calc.\mathfrak{C} \leftarrow []$
8:     $probSelect \leftarrow genProbSelect(Reps)$
9:     $randomSeed \leftarrow commit(randomSeed)$
10:    while $ctr < n_{comp}$ do
11:        $i \leftarrow 0$
12:        $sumReps \leftarrow probSelect[i]$
13:        while $randomSeed > sumReps$ do
14:            $sumReps \leftarrow sumReps + (probSelect[i++] * (2^{2}56))$
15:        if $\neg(i \in calc.\mathfrak{C})$ then
16:            $calc.\mathfrak{C}.append(i)$
17:            $ctr \leftarrow ctr + 1$
18:        $randomSeed \leftarrow commit(randomSeed)$
```

Algorithm 3 Reputation Management

```
19: function $rateComputations(calc, n_{reward}, compDecKey, f_{\varkappa})$
20:     $\mathfrak{C}_{good} \leftarrow []$
21:     $results \leftarrow decrypt(calc.responses, compDecKey)$
22:     $\varkappa \leftarrow f_{\varkappa}(results)$
23:     for $i \in [1, ..., n_{reward}]$ do    ▷ add the $n_{reward}$ closest results to $\varkappa$ to $\mathfrak{C}_{good}$
24:         $\mathfrak{C}_{good}.append(result.\mathcal{C}) \wedge results.remove(result)$ with $distance(result, \varkappa) = min(distance(results, \varkappa))$
25:     $\mathfrak{C}_{bad} \leftarrow results.\mathcal{C}$    ▷ all results not already removed in the for loop are bad results, not to be rewarded
26:     return $\mathfrak{C}_{good}, \mathfrak{C}_{bad}$

27: function $updateReputations(\mathfrak{C}_{good}, \mathfrak{C}_{bad}, calc)$
28:     $avgRepChange \leftarrow length(\mathfrak{C}_{good})/(length(\mathfrak{C}_{good}) + length(\mathfrak{C}_{bad}))$
29:     $denominator \leftarrow \sum(Reps) - (length(Reps) - 1) \cdot (initalRep - 1)$
30:     for $blockProposer \in calc.proposers$ do    ▷ in-line with the results from Section 4.4.2, block proposers rep. changes should be done before updating computers
31:         $Reps[blockProposer] \leftarrow Reps[blockProposer] + (avgRepChange \cdot ((Reps[blockProposer] - (initalRep - 1))/(denominator - Reps[blockProposer])))$    ▷ Necessary for SINCE of proposers/requester
32:     $Reps[calc.requester] \leftarrow Reps[calc.requester] + (avgRepChange \cdot ((Reps[blockProposer] - (initalRep - 1))/(denominator - Reps[blockProposer])))$    ▷ Requester of successfully resolved computation must receive increase in reputation, in line with Section 4.4.2
33:     $Reps[tx.blockProposer] \leftarrow Reps[tx.blockProposer] + (avgRepChange \cdot ((Reps[blockProposer] - (initalRep - 1))/(denominator - Reps[blockProposer])))$    ▷ Proposer including the Finalize transaction must also receive increase in reputation, in line with Section 4.4.2
34:     $Reps[\mathfrak{C}_{good}] \leftarrow Reps[\mathfrak{C}_{good}] + 1$
```

Algorithm 4 Marvel DC smart contract pseudocode

```
35: $\mathfrak{C} \leftarrow []$                                          ▷ Set of active computers
36: $initalRep \leftarrow getInitialRep()$
37: $Reps \leftarrow [initalRep$ for $i \in \mathfrak{C}]$
38: $T \leftarrow getFinalizeDeadline()$                      ▷ Globally-defined finalize deadline
39: $escrow_{comp}, escrow_{req} \leftarrow getEscrows()$ ▷ Globally-defined escrow amounts, in line with Section
    4.4
40: $n_\psi \leftarrow getMinNumComputersPerComputation()$   ▷ Set $n_\psi$ in-line with requirements from
    Section 4.4
41: $tFunctions \leftarrow getTargetFunctions()$ ▷ Define allowable target functions. In reality, this can be
    updated during the protocol

42: Register
43: upon $\langle REGISTER \rangle$ from $\mathcal{P}$ with $\mathcal{P} \notin \mathfrak{C} \wedge \mathcal{P}.balance > escrow_{comp}$ do   ▷ add computer to
    the system
44:     $\mathcal{P}.transfer(escrow_{comp}, contract)$          ▷ Registration cost to prevent Sybil attacks
45:     $\mathfrak{C}.append(\mathcal{P})$
46:     $Reps.append(initalRep)$

47: Request
48: upon
    $\langle REQUEST, calc, n_{comp}, n_{reward}, compEncKey, Reward_i, f_\varkappa \rangle$ from $requester$ with $n_{comp} >$
    $n_\psi \wedge requester.balance > ((n_{reward}, \cdot Reward_i) + escrow_{req}) \wedge f_\varkappa \in tFunctions$ do
49:     $requester.transfer((n_{reward}, \cdot Reward_i) + escrow_{req}, contract)$  ▷ Transfer total reward plus
    requester escrow to contract
50:     $calc.\mathfrak{C} \leftarrow selectComputers(genRandom(), n_{comp})$      ▷ Select computers for computation
51:     $responses \leftarrow []$
52:     $proposers \leftarrow []$       ▷ Array of the players who recorded each $\langle RESPONSE, * \rangle$ transaction
53:     $start \leftarrow Blockchain.height$                          ▷ Record current height of blockchain
54:     $step \leftarrow computing$

55: Response
56: upon $tx \leftarrow \langle RESPONSE, calc, result \rangle$ from $\mathcal{C} \in calc.\mathfrak{C}$
    with $calc.step = computing \wedge Blockchain.height < calc.start + T$ do
57:     $calc.responses.append(result)$       ▷ $result$ should be the computer $\mathcal{C}$'s result of computing
    $calc$, encrypted using $calc.compEncKey$
58:     if $tx.blockProposer \in \mathfrak{C}$ then
59:         $calc.proposers.append(tx.blockProposer)$

60: Finalize
61: upon $tx \leftarrow \langle FINALIZE, calc, compDecKey \rangle$ from $calc.requester$
    with $valid(compDecKey, calc.compEncKey) \wedge \big((calc.step = computing \wedge Blockchain.height$
    $< calc.start + T \wedge length(calc.responses) = calc.n_{comp}) \vee (calc.step = computing$
    $\wedge Blockchain.height \geq calc.start + T)\big)$ do
62:     $calc.transfer(escrow, calc.requester)$            ▷ Returns the escrow to the requester
63:     $calc.proposers.append(calc.requester)$              ▷ Required for SINCE of requester
64:     if $tx.blockProposer \in \mathfrak{C}$ then              ▷ Required for SINCE of proposers
65:         $calc.proposers.append(tx.blockProposer)$
66:     $calc.step \leftarrow finalized$
67:     $\mathfrak{C}_{good}, \mathfrak{C}_{bad} \leftarrow rateComputations(calc, calc.n_{reward}, compDecKey, calc.f_\varkappa)$      ▷ Function
    which deterministically evaluates the goodness of returned computations, returning the indices
    of good and bad computers
68:     $calc.transfer(calc.Reward_i, \mathfrak{C}_{good})$
69:     $updateReputations(\mathfrak{C}_{good}, \mathfrak{C}_{bad}, calc)$
```

Chapter 5

FairTraDEX

This chapter is based on the paper *FairTraDEX: A Decentralised Exchange Preventing Value Extraction* [77].

5.1 Introduction

One of the most prominent and widely-used classes of protocols being run on smart-contract enabled blockchains is that of DEX protocols. DEX protocols allow a specific set of players, whom we call *users*, to exchange one token for another in the presence of MMs, who provide liquidity to users, usually in exchange for a fee. Interacting with a blockchain-based DEX requires a user or MM to first interact with the players who add transactions to the blockchain, known as miners or block producers. These interactions typically reveal a player's intention to trade to the block producer before the transaction is confirmed on the blockchain, and in doing so, present the block producer with what has become known as a miner-extractable value (MEV) opportunity. MEV, first coined in [43], refers to any expected profits the miner of a block can extract from other players interacting with the blockchain. This extraction is performed by manipulating the ordering of, injecting, and/or censoring transactions in prospective blocks. This has been generalised to expected extractable value (EEV) [61], as non-miner players can also perform many of these attacks. Their definition of EEV can be translated as follows, using the terminology of this thesis: *The expected extractable value EEV_i, describes the total value in value units, which is transferred to player $\mathcal{P}_i$ in expectation using a certain strategy which produces* *a transaction, sequence of transactions, or blocks that later become part of the main chain with some probability.*

A significant advancement in DEX protocols was the advent of AMMs, with Uniswap [102] being the most prominent of which. Projects like Flashbots [50] (a direct spin-off to [43]) have identified that AMMs are the main source of recorded EEV ($> 98\%$, as seen in the chart labelled "Extracted MEV Split by Protocol" in [50], of the \$665M in EEV identified by Flashbots since August, 2020). A peer-reviewed analysis in [93] identified \$540.54M in extracted value up to August 2021, indicating the current number provided by Flashbots is significantly lower than the actual amount of extracted value being extracted from DEX protocols. In [74], it has been further highlighted that in Uniswap V3, liquidity providers are losing more to EEV attacks (impermanent loss in that case) than they are collecting in fees. It is clear that the long-term viability of existing DEX protocols is not plausible.

Although many attempts have been made academically to address this source of EEV [22, 40, 62, 51, 15], no satisfactory solution has been found. The protocols presented in these works remain vulnerable to basic EEV-extraction strategies, as outlined in Section 5.2. Therefore, there is a clear gap, both in literature and in practice, to provide a DEX protocol which definitively eliminates all sources of EEV. In this chapter, we provide such a protocol.

5.1.1 Our Contribution

We introduce width-sensitive frequent batch auctions (WSFBAs), idealised commit-reveal exchange protocols between and MMs, based on FBAs [33]. WSFBAs are an important improvement on basic FBAs with respect to decentralised systems. WSFBAs ensure submit market orders in the presence of monopolistic MMs, compared to a standard FBA in which are required to submit limit orders at their interpretation of the true price of the swap, plus some trade fee. The requirement for to submit limit orders leads to worse order execution as trade probability is decreased, while also placing a significant burden on to track this market price. This burden is removed in a WSFBA, providing an "obvious optimal" for users, as coined in [96]. Furthermore, in the case of competing MMs, a WSFBA provides equivalent equilibria to [33].

We describe FairTraDEX, a blockchain-based implementation of a WSFBA. At a high-level, must first register to participate in the protocol by depositing an *escrow* to the smart-contract. To then enter a WSFBA in the FairTraDEX protocol, commit to an order along with a ZK proof proving that they deposited an escrow to the smart contract. MMs simultaneously commit to markets (bids

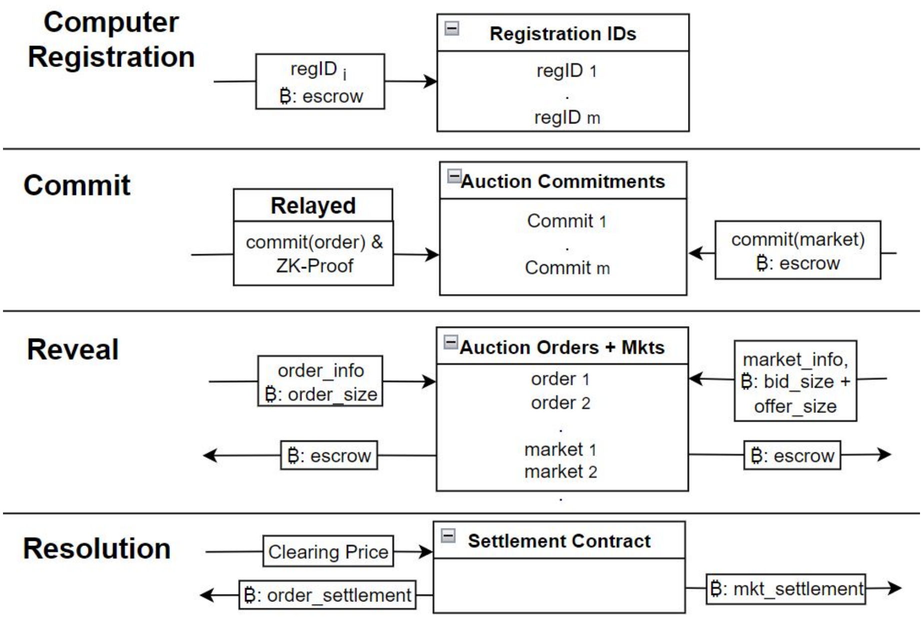

Figure 5.1: FairTraDEX phases before order settlement. ₿ : indicates the transfer of some tokens, but not necessarily in the same denomination.

and offers), although depositing an escrow at time of commitment (MM escrows do not reveal direction due their neutral directional nature). In the proceeding reveal phase, to ensure the correct revelation of information as in the reveal phase of FairTraDEX, any user/MM who committed to orders/markets in the commit phase but do not reveal lose access to their deposit. After the reveal phase finishes, the auction is settled at a single clearing price which maximises trade volume. A representation of the information and escrow flow in FairTraDEX prior to order settlement is presented in Figure 5.1.

In FairTraDEX, order commitments are recorded on-chain (to enforce the corresponding escrow punishment). We utilize ZK set-membership proofs to allow players to commit to their orders anonymously. As such, in FairTraDEX, every user must initially register to the protocol, depositing an escrow. Then, whenever a user wants to commit to an order, the user only has to prove membership of the player set registered in the protocol. Given enough registrations, the probability that a user's ZK set-membership proof/committed order relates to the actual order contents approaches 0 (we formalize this notion in Section 5.5). In other words, no other player in the system can see the committed order and use it to infer anything about what the order is. To definitively hide order information, orders are committed, including the ZK-proof, by using a relayer, a third-party who receives a fee for including relayed transactions in the blockchain (see Section 2.4 for further details).

This combination of tools serves as a subtle, yet crucial, improvement to previous attempts to implement blockchain-based FBAs [22, 51, 40]. In these previous attempts, on-chain order commitments reveal user/order-specific information which allows observant counterparties to improve their ability to guess user order information. With sufficiently many registered users, no counterparty using FairTraDEX can learn information that can be used to bias prices against users with positive expectancy. As such, EEV is prevented in FairTraDEX when enough users register to the protocol.

We provide an extensive Ethereum virtual-machine compatible proof-of-concept for FairTraDEX [54] including a comparison of protocol running costs with previous solutions in Section 5.8, which remain constant with respect to order size, price and direction. When compared to potentially percentage-point slippages and EEV-attack costs required to trade on current DEXs, also highlighted in Section 5.8, FairTraDEX's formal guarantees of protocol-level EEV prevention and up-front, fixed and explicit costs set a new standard for DEX protocols.

5.1.2 Organization of the Chapter

Section 5.2 analyzes previous work related to the construction of EEV-proof DEX protocols. Section 5.3 outlines the terminology used in the chapter, including chapter-specific player definitions needed to formally reason about FairTraDEX. Section 5.4 defines the ideal WSFBA functionality, and identifies the strategies of both users and MMs. Section 5.5 maps the ideal WSFBA functionality to a series of algorithms which form the FairTraDEX protocol. Section 5.6 outlines the properties of FairTraDEX, proving that rational players follow FairTraDEX, and then that FairTraDEX implements a WSFBA. Section 5.7 contains detailed discussions and considerations with regard to implementing FairTraDEX. Section 5.8 provides an analysis of the smart-contract encoding of FairTraDEX, including a cost-benefit comparison to existing solutions. We conclude in Section 5.9.

5.2 Related Work

In this section, we detail the most closely related works to FairTraDEX. The main works aimed at protecting DEX users from EEV either focus on preventing front-running of orders [39, 22], the fair ordering of transactions based on their delivery time [62], or on hiding user trade information until the trade has been committed to the blockchain [51, 15, 40]. Of these works, the closest to our proposal are [39, 22, 51, 40]. All of these works critically depend on honesty from MMs, auction operators and/or the block proposers.

In this chapter we adopt the concept of *expected extractable value* (EEV) as introduced in [61]. There, the authors attempt to formalise extractable value and generalise it beyond value extractable be miners. We also believe it is necessary to model the decision of all rational players based on that can generated by particular orderings of transactions/blocks by any player in the system, and not just the miner. The approach taken is to consider EEV as the maximum of all non-protocol strategies, with protocols considered secure if the EEV of following the protocol is strictly dominated by following the protocol strategy, as formalized in Chapter 3. In this thesis, we also consider an additional case of EEV not necessarily considered in [61] which is prevalent in commit-reveal protocols such as [39]. In such protocols, honest behaviour usually involves sending a valid second transaction (the reveal transaction in a commit-reveal protocol), but where players can extract value in expectancy by not sending these transactions. However, we believe the definition of EEV in [61] can be extended to include these attacks. As such, we also attempt to move away from the legacy use of MEV, and focus instead on the prevention of the more general EEV.

In [39], users submit orders to a single (monopolistic) MM in an off-chain Σ-protocol. Users contact the MM with a size, direction (communicated on-chain) and price (communicated off-chain). If the MM accepts, the MM then publishes the trade on the blockchain, otherwise aborts. User orders are sequentialized so only one order can be executed at a time, preventing the MM from reordering user orders. Crucially, the MM is always allowed to see orders from users and can choose to abort them. It is argued that MMs are happy to trade against all orders, including informed orders. Furthermore, it is assumed that users are independent, with random information. This is not true in real-world trading environments, and as such, Theorem 3 therein may not hold in practice. The paper references [44] as justification for the quality of price/service provided by the MM, however there is a subtle but crucial difference between the games in [44]: in [44] the user has the final decision on whether or not to execute the trade, while in [39] the MM has the final decision. This optionality has an implicit cost for the user and provides a source of EEV to the MM. In FairTraDEX, no optionality is given to the MM or users, while given enough registered users, the direction of any individual user remains hidden until the trade has been committed. As such, FairTraDEX is able to benefit from the results of [44] in a single MM game, that is, liquid (tradeable and of a user-specified width) markets centred around the pre-trade external market price when a trade is accepted by the MM.

The P2DEX protocol [22] is an off-chain MPC protocol run by servers where players can submit orders to exchange tokens from one blockchain to another (although it also appears applicable to one blockchain with many tokens). The orders are encrypted using a threshold secret-sharing scheme with each server receiving a unique share. The protocol has mechanisms to identify double-spending of player funds sent to the servers, and deviation (failure/ misbehaviour) of servers, as the MPC matching protocol is publicly verifiable. As such, all players in the blockchain can verify that a set of orders have been matched correctly, or some of the servers deviated from the protocol. The exchange depends on all servers participating in a secret-sharing protocol to match orders, with at least one server being honest/not colluding with other servers. As with [51, 40], an emphasis is placed on not revealing unmatched orders.

In P2DEX, users must publicly deposit the tokens with which to trade in the same time frame as the order matching takes place, exposing users to standard identity- and directional-based EEV exploits. Separating token deposit and identity revelation from a user's commitment to a specific auction are important advancements used in FairTraDEX to protect against EEV. Furthermore, at least one of the servers in charge of fairly executing orders is required to be honest-by-default. If the servers, a minority subset of players in the P2DEX protocol, are rational and monopolised/colluding, the servers can front-run orders. In FairTraDEX, such value-extraction is prevented by keeping all order information hidden until every order in a particular settlement round has been committed. Furthermore, the set of servers act as a point-of-failure as server participation is required to finalise order-settlement. No subset of players in FairTraDEX can prevent the matching of correctly-revealed orders, while in P2DEX, any majority of servers can prevent order-matching.

Similar commit-reveal protocols to FairTraDEX for blockchain-based token-exchange are proposed in [51, 40]. The protocol in [51] attempts to implement an FBA, and as such has many similarities to FairTraDEX. As in FairTraDEX, the protocol progresses in rounds of Commit, Reveal and Resolution phases. In [51], there is a designated *operator* who is in charge of settling the auction. Players commit to orders in the Commit phase, as well as providing cryptographic information, which is used to prove correct settlement in the Resolution phase. Unlike FairTraDEX, these commit messages are sent by players directly to the blockchain, revealing identity and trade direction. In the Reveal phase, players encrypt their orders using the operator's public key, and send the encryptions to the operator. In the Resolution phase, the operator then chooses a clearing price which intersects the buy and sell liquidity, maximising the notional to be traded. The operator then publishes a list of all matched orders to the blockchain, along with a range proof which is used to verify the correct execution of orders, while not revealing any information about unexecuted orders other than that already revealed in the commit phase. This protocol is exposed to several game-theoretic exploits which contradict its protection against front-running. These include the necessity to reveal order direction a-priori, and the non-trivial handling of the linkability between commitments and account-balances. The protocol also depends on an operator who does not participate in token-exchange, gains exclusive access to order information, and is depended on for protocol completion.

In [40], the protocol attempts to implement an FBA, and is the most similar to FairTraDEX. It is an improvement on [51], with a direct comparison of the two protocols forming the main basis of the justification of [40]. As in [51], the protocol is overseen by an operator who is in charge of receiving orders privately from players and correctly executing the auction. As in FairTraDEX, the protocol progresses in rounds of Commit, Reveal and Resolution phases. In the Commit phase, players commit to orders and publish these commitments to the blockchain. Although not revealing the trade direction as is the case in [51], these commit messages are sent by players directly to the blockchain, and as such reveal identity In the Reveal phase, players encrypt their orders using the operator's public key, and send these encryptions to the operator. The operator then publishes all executed orders, while revealing nothing about unexecuted orders. The validity of execution depends on all players who submitted orders verifying that their order should not have been executed given the list of executions. As users commit their own orders to the blockchain, revealing their identities, this also reveals corresponding token balances and execution patterns which can be used by a basic professional MM to skew prices and extract value from the user. Both of [51, 40] aim to protect unmatched orders from being revealed, but to do so, both protocols depend on a TTP selected before the auction begins to execute order-matching (no one else in the ecosystem can finalise the auction, a single point of failure). Both protocols assume that the operator does not reveal unexecuted order information.

5.3 Preliminaries

This section introduces the terminology and definitions necessary to understand the main results of the chapter. Much of the terminology used in this chapter is introduced in Section 2.5, although we extend this with a detailed definition of an FBA and its main properties, which is specific to this chapter.

In creating a DEX protocol, an idealised goal would be to ensure that there exists an SNE in which users trade at the external market price in expectancy. However, this is unrealistic as MMs are a key component in liquidity provision. Therefore a more realistic, yet still desirable, goal would be to ensure that there exists an SNE where users can trade at the external market price in expectancy in exchange for some pre-determined fee, payable to the MMs, which is bounded by the users' utility gain from the swap. In existing AMMs and DEX protocols, this realistic goal remains unachieved, as explained in Section 5.2. FairTraDEX however, achieves this goal.

Note that MMs differ from liquidity pools in AMMs. The decision logic of AMM liquidity pools is public and deterministic, and any adjustments to liquidity pools must be queued publicly in the mempool, exposing it to EEV attacks. MMs, however, make private trading decisions and communicate them on-chain. One possible action is to add liquidity to an AMM, or in the case of FairTraDEX, add a market to an auction. Following the analysis of [74] and the losses being incurred by liquidity providers in AMMs, players currently providing liquidity in AMMs, although acting honestly, do not fit our rational player model. In FairTraDEX, by ensuring following the protocol forms an SNE, honesty and rationality are equivalent. If players deviate from the protocol in FairTraDEX, this strictly decreases their expected utility, which is further discussed in Section 5.7.1.

5.3.1 Frequent Batch Auctions

As stated in Section 5.1, FairTraDEX is based on an FBA [33]. FBAs are used in many of the largest centralised exchanges such as the FCA[1], CBOE[2], and ESMA[3]. As FBAs were initially intended for a centralised setting, we consider them being run by a TTP who enforces the correct participation of all players. In FairTraDEX, the key TTP functionalities needed to instantiate an FBA are

[1]FCA https://www.fca.org.uk/publications/research/periodic-auctions
[2]CBOE https://www.cboe.com/europe/equities/trading/periodic_auctions_book/
[3]ESMA https://www.esma.europa.eu/sites/default/files/library/esma70-156-1035_final_report_call_for_evidence_periodic_auctions.pdf

replicated using ZK set-membership proofs, incentivization and a blockchain as a censorship-resistant bulletin-board. We define an FBA here using the terminology of the thesis.

Definition 5.3.1. *A frequent batch auction (FBA) (sometimes referred to as a periodic auction) involves users and MMs privately submitting either limit or market orders to the TTP. These orders are collected until a specified deadline. After this deadline, the orders are settled at the clearing price. A single clearing price is chosen which maximises the total notional traded based on the specified sizes and prices of all orders. If there is more supply (quantity of tokens being sold) at the clearing price than demand (quantity of tokens being bought), all sell orders offered at the highest price at or below the clearing price are pro-rated based on size such that supply equals demand at the clearing price. Similarly, if there is more demand than supply at the clearing price, all buy orders bid at the lowest price at or above the clearing price are pro-rated based on size such that demand equals supply at the clearing price. Any limit buy orders below/sell orders above the clearing price are not executed.*

There are two key differences between this definition and the specification in [33]:

1. In our definition, if an order is not fulfilled, it is revealed with any tokens not being sold returned to the seller. This does not affect the game-theoretic guarantees, as the results in [33] only depend on the hiding of order information while players are submitting orders to the auction.

2. As every order in our auction must be submitted independently for each auction, there is no time priority applied when pro-rating orders in case of a supply-demand imbalance. This is a sub-case of the FBAs as defined in [33], and consequently, our protocol retains the same game-theoretic guarantees.

Game-Theoretic Guarantees of a Frequent Batch Auction

In this section we investigate the properties of an FBA between rational MMs and rational users, where MMs do not know the desired trade direction of the users.

We first restate, using the terminology from this thesis, the main result from [33] which applies to our game-theoretically equivalent definition of an FBA. To do this, we let D represent the net trade imbalance of users in a particular instance of an FBA in terms of ₿. A positive D indicates a user buy imbalance (more user buyers than sellers of the swap), while a negative D indicates a user sell imbalance. We require a finite bound on the absolute imbalance, which we denote $Q_{not} < \infty$, for the existence of optimal MM strategies. As in [33], we assume that $|D| \leq Q_{not}$, and in-keeping with the notion of an external market price, D is symmetric around 0 at the external market price.

Theorem 5.3.2. *[33] For an FBA with at least two non-cooperative MMs, there is a strict Nash equilibrium where users only submit market orders and MMs show a market of width 1 (bid = offer) centred around the external market price in size greater than Q_{not}.*

This is a useful result in the case of at least two non-cooperative MMs, with users receiving a game-theoretic guarantee that they can exchange one token for another at the external market price in expectancy in an FBA. Furthermore, as MM liquidity is greater than the net user trade size, the implicit impact to these trades in [33] is bounded by the width, which is 1. As users have a strictly positive utility for exchanging tokens, this is equivalent to users always having positive expectancy to participate in an FBA. However, it is also shown in this equilibrium that MMs have 0 expected utility. A basic adjustment to the protocol in that setting would then be to charge users a fee for the service and pro-rate these fees to the MMs to ensure the long-term participation of MMs.

5.4 Width-Sensitive Frequent Batch Auctions

In this section we outline the properties of an idealised variation of an FBA which we define as a width-sensitive FBA. Width-sensitive FBAs maintain the desirable properties of FBAs with respect to optimal strategies for MMs and users [33], while also adding important protections for users in a decentralised setting where monopolistic MMs may exist. The important assumption with regard to the guarantees of an FBA is the presence of at least two non-cooperative MMs. In a decentralised setting, this can be seen as insufficient. One of the most desirable properties of FBAs in the presence of 2 non-cooperative MMs is the fact that users submit market orders. We envisage users as relatively uninformed players for whom choosing the correct/fair price to trade has an implicit cost. Market orders remove this burden, providing users with an "obvious optimal" as advocated in [96]. To reach a similar equilibrium in the presence of a monopolistic MM, we must amend the basic FBA protocol. In this section, we define a *width-sensitive* FBA (WSFBA) to handle monopolistic MMs, while retaining the desirable properties of an FBA in the presence of two or more non-cooperative MMs.

In the presence of a single rational MM, we need to utilise the value gained by users for exchanging token. That is, recall from Section 5.3, users in our protocol observe a positive utility of at least the minimum user fee *fee* for exchanging tokens. In a WSFBA, this fee is translated to a market width, and input with users orders as a maximum market width on which users are willing to trade. This allows us to prove submitting market orders remains a SNE. Conversely in an FBA, if MMs cooperate/are replaced by a monopolistic MM, submitting market orders is a strictly dominated strategy for users, with users now required to submit a limit price. WSFBAs avoid this degradation of user experience, and the corresponding reduced probability of execution and quality of liquidity this has on FBAs.

We now define a WSFBA.

Definition 5.4.1. *A width-sensitive frequent batch auction (WSFBA) involves MMs submitting markets to the TTP with total notional on the bid and offer of at least Q_{not}. Users and MMs privately submit limit and market orders to the TTP including a requested maximum width from the tightest MM, above which the order is not executed. Orders are collected until a specified deadline. After this deadline, user orders with requested width greater than or equal to the tightest MM width, along with a randomly-selected market from the tightest provided markets, are settled at a single clearing price which maximises the total notional traded, and then minimises the net trade imbalance.*[4] *If there is more supply at the clearing price than demand, sell orders at the highest price at or below the clearing price are pro-rated based on size such that supply equals demand at the clearing price. Similarly, if there is more demand than supply at the clearing price, buy orders at the lowest price at or above the clearing price are pro-rated based on size such that demand equals supply at the clearing price. Any limit buy orders below/sell orders above the clearing price are not executed.*

The key differences between a conventional FBA and a WSFBA are the specification of MM widths by users, the minimum MM notional requirement on the bid and offer, and the requirement for the clearing price to minimise the imbalance over all prices which maximise the notional traded. Minimising imbalance is a small optimisation which produces a reasonable and precise clearing price when MMs do not show width 1 markets as in an FBA. A precise algorithm

[4]As Q_{not} is greater than the absolute user order imbalance, the clearing price must lie between the MM bid and offer

for verifying a given clearing price satisfies these proprieties is included both in Algorithm 8 and [54], and described in Section 5.6.1. The other amendments are intended to protect users against monopolistic MMs, and are discussed in the proceeding section.

5.4.1 Properties of Width-Sensitive Frequent Batch Auctions

In Theorem 5.4.3, we identify an SNE for WSFBAs, and show that it is equivalent to the SNE of an FBA. The case of a single monopolistic MM is more complex. First, we observe that an MM in a WSFBA always shows a market with reference price equal to the external market price.

Lemma 5.4.2. *For an MM in a WSFBA between x and y with external market price equal to* $\varepsilon_{x\to y} = \frac{\varepsilon_y}{\varepsilon_x}$ *and a user order of notional* $Z\text{฿} > 0$*, she strictly maximizes her expected utility by showing a market with reference price* $\varepsilon_{ref} = \varepsilon_{x\to y}$ *for any fixed width* $width \geq 1$.

Proof. Let us define the market in terms of ε_{ref} and *width* as described in Section 5.3, namely, $\frac{\varepsilon_{ref}}{\sqrt{width}}$ @ $\sqrt{width}\,\varepsilon_{ref}$. In the cases of a user buyer and user seller of the swap, we convert user notional orders into the tokens being sold, mark the trades to their respective external market prices using a multiplicative market-impact coefficient for the token swap of δ, then reconvert the tokens into notional.

If the user is a buyer of the swap, the user is selling x, with trade size of $\frac{Z\text{฿}}{\varepsilon_x}$ in x. The trade occurs on the token swap offer of $\sqrt{width}\,\varepsilon_{ref}$, resulting in the sale of $\frac{Z\text{฿}}{\varepsilon_x}\frac{1}{\sqrt{width}\,\varepsilon_{ref}}$ token y by the MM. Finally, the y bought by the user have an expected per-token value of $\sqrt{\delta}\varepsilon_y$, while the notional acquired by the MM (x) has an expected value of $\frac{Z\text{฿}}{\sqrt{\delta}}$. This is an expected net profit for the MM measured in notional of:

$$\frac{Z\text{฿}}{\sqrt{\delta}} - \frac{Z\text{฿}}{\varepsilon_x\sqrt{width}\,\varepsilon_{ref}}\sqrt{\delta}\varepsilon_y = \frac{Z\text{฿}}{\sqrt{\delta}} - \frac{\sqrt{\delta}Z\text{฿}}{\sqrt{width}}\frac{\varepsilon_{x\to y}}{\varepsilon_{ref}}. \tag{5.1}$$

If the user is a seller of the swap, the user is selling y, with trade size of $\frac{Z\text{฿}}{\varepsilon_y}$ in y. The trade occurs on the token swap bid of $\frac{\varepsilon_{ref}}{\sqrt{width}}$, resulting in the sale of $\frac{Z\text{฿}}{\varepsilon_y}\frac{\varepsilon_{ref}}{\sqrt{width}}$ token x by the MM. Finally, the x bought by the user have an expected per-token value of $\sqrt{\delta}\varepsilon_x$, while again, the notional acquired by the MM (y) has an expected value of $\frac{Z\text{₿}}{\sqrt{\delta}}$ This is an expected net profit for the MM of:

$$\frac{Z\text{₿}}{\sqrt{\delta}} - \frac{\sqrt{\delta}Z\text{₿}}{\sqrt{width}}\frac{\varepsilon_{ref}}{\varepsilon_{x\to y}}. \tag{5.2}$$

We know the expected buying and selling of Z₿ notional at $\varepsilon_{x\to y}$ are equally likely by the definition of an external market price as a perfectly-informed signal. Therefore, the total expected profit is:

$$Z\text{₿}\Big(\frac{1}{2}(\frac{1}{\sqrt{\delta}} - \frac{\sqrt{\delta}}{\sqrt{width}}\frac{\varepsilon_{x\to y}}{\varepsilon_{ref}}) + \frac{1}{2}(\frac{1}{\sqrt{\delta}} - \frac{\sqrt{\delta}}{\sqrt{width}}\frac{\varepsilon_{ref}}{\varepsilon_{x\to y}})\Big). \tag{5.3}$$

To find the maximum with respect to ε_{ref}, we take the first derivative of this formula, and let it equal to 0:

$$\frac{\varepsilon_{x\to y}}{\varepsilon_{ref}^2} - \frac{1}{\varepsilon_{x\to y}} = 0. \tag{5.4}$$

Solving for ε_{ref} gives $\varepsilon_{ref} = \varepsilon_{x\to y}$, which is equivalent to the MM strictly maximizing her expected profits by letting $\varepsilon_{ref} = \varepsilon_{x\to y}$. □

This result is independent of the choice of width and market-impact coefficient. However, it assumes that the MM trades with the user on either the bid or the offer. With respect to a WSFBA without notional restrictions and a monopolistic MM, if users submit market orders, there are fringe cases (large imbalances) which incentivize MMs to show markets far from the external market price. Removing these restrictions from a WSFBA makes for interesting future work.

Recall users have a strictly positive utility to exchange tokens described by the minimum user fee *fee*, which is equivalent to being strongly incentivized to trade on a market with reference price ε_{ref} and width $width \leq fee$. With this in mind, we can now apply the main result of [33] to a WSFBA.

Theorem 5.4.3. *For a WSFBA, the strict Nash equilibria strategies given the number of non-cooperative MMs submitting markets being N are:*

- $N = 1$*: users submit market orders of requested width fee and the MM shows a market of width at most fee with reference price equal to the external market price.*

- $N \geq 2$*: users submit market orders of requested width greater than 1 and MMs show a market of width 1 with reference price equal to the external market price.*

Proof. We now investigate each of the cases described in the theorem statement in terms of the number of non-cooperative MMs N.

$N = 1$: Consider first the strategy of a user. Although users are not necessarily aware of the external market price, let us consider their strategies taking the external market price *price* as a variable with an arbitrary distribution. For buy orders, the strategy of submitting a limit order with price *price* less than $\sqrt{fee}\,\varepsilon$ is dominated by all prices greater than *price* and less than or equal to $\sqrt{fee}\,\varepsilon$. For limit sell orders, this limit is $\frac{\varepsilon}{\sqrt{fee}}$. As such, the equilibrium for users involves submitting orders equivalent to a market of width equivalent to at least *fee* with reference price equal to the external market price. If a user knows a MM submits a market of width less than or equal to *fee* with reference price equal to the external market price, this strategy is further dominated by submitting a market order with requested width *fee*, as market orders strictly increase the user's probability of trading. Furthermore, any strategy for a user which involves trading on a price outside $[\frac{\varepsilon}{\sqrt{fee}}, \sqrt{fee}\,\varepsilon]$ is strictly dominated by not trading. As such, the only possibilities for equilibria can occur on a market of $\frac{\varepsilon}{\sqrt{fee}}@\sqrt{fee}\,\varepsilon$. Furthermore, the submission of market orders (increasing probability of trading) with requested width *fee* is strictly dominant if the MM shows a market with reference price equal to the external market price in sufficient size to fill all users' orders. As $Q_{not} > |D|$, this would be the case given the appropriate reference price. In Lemma 5.4.2, we have seen that a MM trading on a market against a random user shows a market with reference price equal to the external market price. Therefore users submit market orders with requested width *fee*. Moreover, this strategy does not require the user to know the external market price.

Consider now the MM strategy. As only the tightest market in every auction is included for settlement, the MM only submits one market. Any MM order bidding above/offering below the external market price has negative expected utility, and as such, no rational MM does this this. Also, by definition, the MM must show a market in size $Q_{not} \geq |D|$, meaning the MM has sufficient notional on the bid and offer to trade all user orders and as such the clearing price must be inside the provided MM market (between the bid and offer prices). Next, we have seen in Lemma 5.4.2 that a MM trading on a market against a random user shows a market with reference price equal to the external market price. Furthermore, from Equation 5.3 we can see that the expected utility of a MM is strictly increasing in width. Any strategy involving a market with width greater than *fee* cannot be an equilibrium as users strictly prefer to trade on markets of lesser width, as argued above. Therefore, the MM maximises her expectancy against a random user by showing a market of width *fee* with reference price equal to the external market price. Against multiple users, a positive notional imbalance at the external market price is decreasing in price (resp. a negative notional imbalance is increasing as price decreases), which may cause the MM to provide a market of width less then *fee*.

Consider the strategy of a MM providing a market of width less than or equal to *fee* with reference price equal to the external market price, and the strategy of users submitting market orders with requested width *fee*. We have shown that any player deviation strictly decreases that player's expectancy, making this a strict Nash Equilibrium.

$N \geq 2$: As MMs in a standard FBA provide markets of width 1 when the width is not a restriction, applying the requested width adjustments of a WSFBA, further incentivizing tighter markets, does not change the unique equilibrium of Theorem 5.3.2. Similarly, as there is a unique clearing price when a width-1 market is submitted, it must also minimise the imbalance over prices that maximise total notional traded. The restriction on the notional of markets in a WSFBA is in line with the inequality of Theorem 5.3.2. Furthermore, any requested width > 1 in this equilibrium ensures a user's order trading through the external market price is included in the final auction settlement, with the maximum allocation occurring when a user submits a market order. □

Theorem 5.4.3 identifies that users always submit market orders, and in settings where it is unclear whether there is a single monopolistic MM, or many non-cooperative MMs, it can be seen that users always submit market orders with requested width *fee*.

5.5 FairTraDEX

In Section 5.4 we constructed a WSFBA using a TTP to enforce correct player balances, order sizes, revelation of orders, correct calculation of the clearing price and the settlement of orders. In a decentralised setting with rational players, such a TTP does not exist. However, we do have access to censorship-resistant public bulletin boards in the form of blockchain-protocols. As discussed in the Section 5.1, these bulletin boards have many caveats such as the ordering of transactions based on transaction send time not being preserved (transaction re-ordering attacks). However, if we are able to bound the delay of updates being added to such a bulletin board (transactions being confirmed on the blockchain), we can implement a WSFBA in such a setting.

In this section we construct the FairTraDEX protocol as a sequence of algorithms. We then provide a series of results regarding the incentive compatibility of these algorithms with the goal of proving FairTraDEX instantiates a WSFBA, and that following the protocol is an SNE.

5.5.1 System Model

1. All players $\mathcal{P}_1, ..., \mathcal{P}_n$ are members of a blockchain-based distributed ledger, and a corresponding PKI.

2. The ledger is represented by a linear blockchain with its state progressing by having new blocks sequentially appended. For simplicity, we assume instant finality of blocks meaning that such an appended (valid) block cannot be replaced at any later point in time.

3. A transaction submitted by a player for addition to the blockchain (either directly or relayed) while observing blockchain height H, is included (and thus finalised) in a block of height at most $H + T$, for some known $T > 0$, given that the transaction remains valid for sufficiently many intermediate ledger states.

4. The public NIZK parameters are set-up in a trusted manner.

We do not make any assumptions regarding transaction ordering in blocks. Specifically, the order in which transactions are executed is at the discretion of the block proposer. Our assumption on block finality helps to simplify the proceeding analysis. The protocol can be extended to blockchains with eventual finality, by increasing T to be the number of blocks between a transaction tx_1 being finalised on the blockchain, and the maximum number of blocks required to finalise another transaction tx_2 which is submitted for addition to the blockchain when tx_1 is finalised.

If block producers are participating as MMs/users, we need to adjust T. Let $0 < m < 1$ bound the fraction of blocks produced over chains of length greater than T by a MM responding to/the set of users requesting trades in a particular instance of a FBA (we need to consider all users in a request phase, as they may all have the same direction, and as such, some positive expectancy to preventing a MM revelation). We need to increase T by a factor of $\frac{1}{1-m}$ (similar to the methodology behind the Chain Quality property in [52, 91]). Moreover, our property can be seen as a 'block-based' variant of the time-based liveness property defined in [52, 91]. An example for instant finality is Algorand [38] which stands in contrast to, e.g., Bitcoin which only guarantees eventual finality, while example of a public NIZK parameter setup is a Perpetual Powers of Tau ceremony, as used in Zcash [89].

5.5.2 FairTraDEX Algorithms

Each player $\mathcal{P}_i$ owns (has exclusive access to) a set of token balances bal_i which are stored as a global variable. For a token tkn, $bal_i(tkn)$ is the amount of token tkn that $\mathcal{P}_i$ owns. Algorithm outputs are not signed, so players observing the output of an algorithm instance can only infer information about the player running the algorithm from public outputs and any corresponding global variable updates.

We now outline FairTraDEX as a set of algorithms: Setup(), Register(), CommitUser(), CommitMM(), RevealUser(), RevealMM() and Resolution(). A FairTraDEX instance is initialised by running Setup(), and proceeds indefinitely in rounds of three distinct, consecutive phases: *Commit*, *Reveal* and *Resolution*, each of length T blocks (see Section 5.5.1). For readability, we provide here the intuition to the algorithms of FairTraDEX, with a detailed pseudocode implementation of these algorithms provided in Algorithms 5-9.

Players in the underlying blockchain protocol can enter FairTraDEX as users by running an instance of Register(), which for a given user deposits an escrow $escrow_{user}$, and generates private information ($\mathcal{S},\ \mathcal{R} \in \{0,1\}^{\Theta(\kappa)}$) which is used in CommitUser() to prove that the user indeed deposited an escrow, without revealing which deposit.

In the Commit phase, all players can run any number of CommitUser() and/or CommitMM() instances. CommitUser() generates a user order, commits to that order publicly and proves in ZK that the player deposited an escrow. If such a proof cannot be generated, or a proof has already been generated for the same $\mathcal{S}$, no order can be committed. A correctly run CommitMM() instance generates a market for a prospective MM, commits to that market publicly and deposits an escrow $escrow_{MM}$.

In the Reveal phase, players can run any number of RevealUser() and/or RevealMM() instances. RevealUser() publishes an order generated through CommitUser(), returning the escrow corresponding to the CommitUser() instance, and as such the Register() instance, to the user. RevealMM() publishes a market corresponding to a CommitMM() instance, and returns the corresponding escrow. Both Reveal phase algorithms assert that the user and MM have sufficient token balances to submit their order and market respectively. These assertions are also ensured in the Commit phase, but must be rechecked to ensure correct balances at the point of token transfer.

In the Resolution phase, any number of Resolution() instances can be run. The first correct Resolution() instance selects the tightest market from the set of revealed markets, *revealedMkts*, for inclusion in order settlement, and any tie-breaks settled using *commit*(*revealedMkts*), as a random seed[5]. The clearing price which maximises notional traded, and then minimises the notional imbalance of the remaining market and orders is computed. A precise algorithm for verifying the clearing price is provided in Algorithm 8 lines 95-118, and described in Section 5.6.1. The intuition behind the algorithm is as follows: Given more tokens are sold than bought at the proposed price, it can be seen that checking the next price point lower, first for higher traded notional, and then for a greater or equal absolute imbalance is sufficient to verify the proposed price is a valid clearing price. The equivalent check at the next price point above holds when more tokens are bought than sold at the proposed price.

Orders and markets are then settled based on this clearing price. Finally, the arrays tracking active commitments, orders and markets *userCommits*, *MMCommits*, *revealedOrders*, *revealedMkts* are cleared, so unsuccessfully revealed commitments during this round cannot be used to run RevealUser() or RevealMM() in future rounds. This effectively destroys the deposited escrows of such commitments.

As mentioned already, FairTraDEX protocol encodings are provided in Algorithms 5-9. In these algorithms, for arrays containing array objects, the array objects are uniquely identifiable by the first item in the array (i.e. User identifier, serial number, ZKProof). For an arbitrary bit-string $m \in \{0,1\}^*$, *relay*(m) indicates broadcasting m to the relay transaction mempool, where m is included as a transaction in the blockchain if and only if it gives the including relayer access to a relayer fee fee_r.

5.5.3 FairTraDEX vs. WSFBA

The main differences between FairTraDEX and a WSFBA are as follows:

- Escrows are used to enforce the correct revelation of players who commit to orders or markets. Escrows are only returned to players if orders are

[5]Given all markets are revealed, the final value of *revealedMkts*, and as such *commit*(*revealedMkts*), is unpredictable in the presence of two or more non-cooperative MMs. We prove in Lemma 5.6.3 that all MMs running CommitMM() also run RevealMM().

revealed and correspond to a valid commit. Furthermore, escrows are chosen large enough to ensure the reclamation of escrows has strictly higher utility than not, ensuring rational players follow the protocol.

- An algorithm involving deposits and/or withdrawals updates the set of balances for all players, identifying the player calling the algorithm.
- FairTraDEX separates the depositing of user escrow and user order commitments. This is a key functionality necessary to preserve user anonymity and the guarantees of a WSFBA. If a user deposits an escrow in the same instance as committing to an order, that information can be used to identify the player, and imply information about the player's order. By separating the two, commitment does not require the update of global variables that can be used to identify the user.
- Set-membership proofs in ZK in the CommitUser() algorithm are used to prove that a player committing to a user order has deposited a user escrow. As FairTraDEX separates the deposit and commitment steps, these proofs allow a user who deposited an escrow to generate one (and only one, as ZK proofs reveal $\mathcal{S}$) order per escrow, while only revealing that the order corresponds to a deposited escrow. As the number of deposited escrows increases, the probability that an order commitment matches any particular escrow approaches 0. This replicates the anonymous order submission of a WSFBA.
- Tokenised incentives are used to ensure some player in the blockchain calculates the clearing price, and settles orders correctly.

5.5.4 Smart Contract Implementation vs. Algorithmic encoding

We outline here the key differences between Algorithms 5-9, and the smart-contract encoding [54]. As a blockchain-based implementation under the model of Section 5.5.1 involves a PKI for message sending, all public algorithm outputs must now be signed using the PKI. These messages must now be included in blockchain transactions, with a transaction fee required to ensure the transaction gets added to the blockchain.

For a player to publish a transaction to a blockchain-based smart contract without revealing her identity, she must utilise a relayer (for details on relayers, see Section 2.4). Otherwise, the transaction fee is payable from her account, revealing sensitive information such as trading patterns and account balances. Furthermore, this relayer must be rewarded on-chain for relaying the transaction. This reward is added by the user when depositing her escrow, and retrievable by the first relayer publishing the transaction to the blockchain. Furthermore all checks, such as those for the previous use of serial numbers in CommitUser(), or the recording of the tightest MM width in Resolution(), are explicitly encoded in the provided implementations.

Algorithm 5 Register

```
1: $players \leftarrow generatePopulation()$
2: $RegIDs \leftarrow []$
3: $userCommits \leftarrow []$
4: $MMCommits \leftarrow []$
5: $phase \leftarrow$
6: $width_{tight} \leftarrow any$
7: $revealedBuyOrders,\ revealedSellOrders, revealedMkts \leftarrow []$
8: $lastPhaseChange \leftarrow 0$    ▷ tracks the block number of last step update
9: $fee_r \leftarrow getRelayerFee()$
10: $minTickSize \leftarrow getMinimumTickSize()$
11: $commitDeadline, revealDeadline \leftarrow T$    ▷ Deadline for responses equal to the maximal reveal delay described in the Threat Model
12: $escrow_{user} \leftarrow \mathfrak{E}$    ▷ Escrow required to show each market, in line with the Threat Model
13: $Q_{not} \leftarrow getMaxAuctionNotional()$
14: $c \leftarrow random(\mathbb{R}_{>1})$
15: $escrow_{MM} \leftarrow c \cdot Q_{not}$    ▷ Escrow required to show each market, some amount greater than $Q_{not}$
16: $\varepsilon_x \leftarrow getTokenAIndicativePrice()$
17: $currAucNotional \leftarrow 0$
18: $blacklistedSNs \leftarrow []$    ▷ Tracks revealed serial numbers that misbehaved

19: function Initialise()
20:     $phase \leftarrow Commit$

21: upon $\langle CLIENT\text{-}REGISTER, regID\rangle$ from $player \in players$
    with $player.balance(₿) > escrow_{user} + fee_r$ do    ▷ register player as a user
22:     $player.transfer(escrow_{user} + fee_r, ₿, protocolContract)$    ▷ Add user deposit to the contract account
23:     $users.append(regID)$
```

5.5.5 Description of FairTraDEX Encoding

The following is an overview of the blockchain-based encoding of FairTraDEX from Algorithms 5-9.

Register

Users randomly generate $\mathcal{S},\ \mathcal{R} \in \{0,1\}^{\Theta(\kappa)}$, and compute *regID* ← *commit*($\mathcal{S}, \mathcal{R}$). Then, the user sends a ⟨*CLIENT-REGISTER*, *regID*⟩ to the blockchain using

Algorithm 6 Commit

```
24: upon $relay(\langle COMMIT, com, \mathcal{S}, \pi\rangle)$ from $player \in players$ with
    $currAucNotional < Q_{not} \wedge$ Verify$(\pi, com) = 1 \wedge phase = Commit \wedge \neg(\mathcal{S} \in blacklistedSNs)$
    do
25:     $currAucNotional \leftarrow currAucNotional + escrow_{user}$
26:     $userCommits.append([\mathcal{S}, com])$
27:     $protocolContract.transfer(fee_r, ₿, player))$          ▷ Reward relayer

28: upon $\langle COMMIT, com\rangle$ from $player \in players$ with $player.balance(₿) > escrow_{MM} \wedge$
    $\neg(player \in MMCommits)$ $phase = Commit$ do     ▷ Allow only one market per player address
29:     $player.transfer(escrow_{MM}, ₿, protocolContract)$  ▷ Transfer escrow to the protocol contract
    account
30:     $MMCommits.append([player, com])$

31: upon $phase = Commit \wedge Blockchain.height() = lastPhaseChange + commitDeadline$ do
32:     $phase \leftarrow Reveal$
33:     $lastPhaseChange \leftarrow Blockchain.height()$
```

the blockchains PKI. Upon addition to the blockchain, this deposits an escrow of $escrow_{user}$ and a relayer fee of fee_r to the blockchain (line 22), with *regID* added to *users* (line 23).

Commit

A user wishing to submit an order of the form $order \leftarrow (tkn, size, price, width)$, first generates a commitment to the order $com \leftarrow commit(order)$. The user then generates a NIZK signature of knowledge $\pi \leftarrow \text{NIZKSoK}(com)\{(\ regID, \mathcal{R}) : \text{MemVerify}(RegIDs, regID) = 1 \ \ \& \ \ regID = commit(\mathcal{S}, \mathcal{R})\ \}$ on the commitment. The user then relays a message of the form $\langle COMMIT, com, \mathcal{S}, \pi\rangle$ to the relayer mempool, which is then sent to the smart contract by a relayer. The transaction is only valid if Verify(π) returns 1, and as such, a relayer cannot tamper with *com*. The contract first checks that the maximum auction notional has not been reached ($currAucNotional < Q_{not}$, line 24).

Furthermore, a valid $\langle COMMIT, com, \mathcal{S}, \pi\rangle$ message must not reveal a serial number $\mathcal{S}$ which has previously been added to *blacklistedSNs* (initialised line 18). Serial numbers in *blacklistedSNs* correspond to user commits that were not revealed according to the protocol during a previous Reveal phase. The escrow corresponding to serial numbers of *blacklistedSNs* are effectively burned by the protocol, with users permanently losing access to them. If $\mathcal{S}$ is not in *blacklistedSNs*, the order commitment is recorded in *userCommits* (line 26), and the relayer who relays the transaction to the blockchain receives the fee (line 27).

Algorithm 7 Reveal

```
34: upon $\langle$CLIENT-REVEAL, $tkn, size, price, width, \mathcal{S}, \mathcal{R}, regID$ , $regIDNew\rangle$
    from $player \in players$ with $\mathcal{S} \in userCommits \wedge regID = commit(\mathcal{S}, \mathcal{R})$
    $\wedge$ $commit(tkn, size, price = userCommits[\mathcal{S}].com \wedge phase = Reveal$ do
35:     if $price \neq withdraw$ then
36:         if $tkn = x \wedge player.balance(x) \geq size$ then
37:             $size \leftarrow minimum(size, escrow_{user}/\varepsilon_x)$
38:             $revealedBuyOrders.append([player, size, price, width])$ ▷ Add user order to array of
    orders to trade
39:             $player.transfer(size, x, protocolContract)$
40:             if $regIDNew = \varnothing$ then
41:                 $protocolContract.transfer(escrow_{user}, ₿, player))$
42:             else
43:                 $users.append(regIDNew)$
44:         else if $tkn = y \wedge player..balance(y) \geq size$ then
45:             $size \leftarrow minimum(size, escrow_{user}/(\varepsilon_x \cdot price))$
46:             $revealedSellOrders.append([player, size, price, width])$ ▷ Add user order to array of
    orders to trade
47:             if $regIDNew = \varnothing$ then
48:                 $protocolContract.transfer(escrow_{user}, ₿, player))$
49:             else if $player.balance(₿) > fee_r$ then
50:                 $player.transfer(fee_r, ₿, protocolContract)$
51:                 $users.append(regIDNew)$
52:     else                                                    ▷ User wants to withdraw
53:         $protocolContract.transfer(escrow_{user}, ₿, player))$
54:     $users.remove(regID)$
55:     $userCommits.remove(\mathcal{S})$

56: upon $phase = Reveal \wedge Blockchain.height() = lastPhaseChange + revealDeadline$ do
57:     $tieBreaker \leftarrow 0$
58:     $tightMkt \leftarrow ()$
59:     $tieBreakSeed \leftarrow commit(revealedMkts)$     ▷ Generate seed for tie-breaks before revealed
    markets is changed
60:     for $MM \in revealedMkts$ do ▷ Select the unique market corresponding to the tie-breaker in
    the proceeding If statement
61:         if $(Q_{not}/(\varepsilon_x \cdot MM.offer) \leq MM.size_{offer} \leq MM.balance(y))$
    $\wedge$ $(Q_{not}/\varepsilon_x \leq MM.size_{bid} \leq MM.balance(x))$ then     ▷ Check MM *still* has provided the
    minimum required liquidity
62:             $protocolContract.transfer(escrow_{MM}, ₿, MM))$
63:             if $width_{tight} = any \vee (\frac{offer}{bid} < width_{tight})$
    $\vee$ $(\frac{offer}{bid} = width_{tight} \wedge commit(tieBreakSeed||MM||MM.market) > tieBreaker)$ then
64:                 $width_{tight} \leftarrow \frac{offer}{bid}$
65:                 $tieBreaker \leftarrow commit(tieBreakSeed||MM||MM.market)$
66:                 $tightMkt \leftarrow [MM, market]$
67:         $revealedMkts.remove(MM)$
68:     $size_{bid} \leftarrow minimum(tightMkt.size_{bid}, escrow_{MM}/\varepsilon_x)$
69:     $size_{offer} \leftarrow minimum(tightMkt.size_{offer}, escrow_{MM}/(\varepsilon_x \cdot tightMkt.offer))$
70:     $revealedBuyOrders.append([player \leftarrow tightMkt.MM,$
    $size \leftarrow size_{bid}, price \leftarrow tightMkt.bid, width \leftarrow any])$   ▷ Add tightest market to set of orders to
    be settled
71:     $revealedSellOrders.append([player \leftarrow tightMkt.MM,$
    $size \leftarrow size_{offer}, price \leftarrow tightMkt.offer, width \leftarrow any])$
72:     $tightMkt.MM.transfer(size_{bid}, x, protocolContract)$
73:     $tightMkt.MM.transfer(size_{offer}, y, protocolContract)$
74:     for $\mathcal{S} \in userCommits$ do ▷ Add all users who did not reveal order to blacklist, preventing
    further commitments
75:         $blacklistedSNs.append(\mathcal{S})$
76:         $userCommits.remove(\mathcal{S})$
77:     for $MM \in MMCommits$ do                         ▷ MMs who did not reveal market in time
78:         $MMCommits.remove(MM)$ ▷ Remove from protocol without adding to *revealedOrders*,
    effectively burning escrow
79:     $phase \leftarrow Resolution$
80:     $lastPhaseChange \leftarrow Blockchain.height()$
```

Algorithm 8 Reveal (continued), Resolution, Clearing Price Verification

```
88: upon $\langle MM\text{-}REVEAL, market \leftarrow (bid,\ size_{bid},\ offer,\ size_{offer})\rangle$ from $MM \in MMCommits$ with $commit(market) = MMCommits[MM].com \wedge phase = Reveal$ do
89:     if $(Q_{not}/(\varepsilon_x \cdot offer) \leq size_{offer} \leq player.balance(y)) \wedge (Q_{not}/\varepsilon_x \leq size_{bid} \leq player..balance(x))$ then    ▷ Check MM has provided the minimum required liquidity, $Q_{not}$
90:         $revealedMkts.append(MM, market)$
91:         $MMCommits.remove(MM)$

92: upon $phase = Reveal \ \wedge\ len(MMCommits) = 0 \ \wedge\ len(userCommits) = 0$ do    ▷ All reveals published
93:     $phase \leftarrow Resolution$
94:     $lastPhaseChange \leftarrow Blockchain.height()$

95: upon $\langle CP, volumeSettled, imbalance\rangle$ from$player \in players$ with $player.balance(₿) > resBounty \wedge phase = Resolution$ do
96:     $player.transfer(resBounty, ₿, protocolContract)$    ▷ To prevent Sybil attacks, player must deposit funds which are returned if $CP$ is valid
97:     $revealedSellOrders \leftarrow revealedSellOrders[revealedSellOrders..width() > width_{tight} \ \vee\ revealedSellOrders.width() = any]$    ▷ Remove sell orders that cannot trade due to requested width
98:     $revealedBuyOrders \leftarrow revealedBuyOrders[revealedBuyOrders..width() > width_{tight} \ \vee\ revealedBuyOrders.width() = any]$
99:     Assert$(volumeSettled > 0 \ \vee\ \text{minimum}(revealedSellOrders.price) < \text{maximum}(revealedBuyOrders.price))$    ▷ If the indicated clearing price is below the lowest offer/above highest bid, all of the proceeding checks pass.
100:    $buyVolume \leftarrow sum(revealedBuyOrders[revealedBuyOrders.price \geq CP].size)$
101:    $sellVolume \leftarrow sum(revealedSellOrders[revealedSellOrders.price \leq CP].size)$
102:    Assert$(\text{minimum}(buyVolume/CP, sellVolume) = volumeSettled))$
103:    Assert$((buyVolume/CP) - sellVolume = imbalance\ )$
104:    if $imbalance = 0$ then    ▷ We are done
105:        $SettleOrders(CP, buyVolume, sellVolume)$
106:    if $imbalance > 0$ then    ▷ As the auction is bid at $CP$, check if next price increment above clears higher volume OR smaller imbalance
107:        $priceToCheck \leftarrow CP + minTickSize$
108:        $buyVolumeNew \leftarrow (buyVolume - sum(revealedBuyOrders[CP \leq revealedBuyOrders.price < priceToCheck].size))/CP$
109:        $sellVolumeNew \leftarrow sellVolume + sum(revealedSellOrders[CP < revealedSellOrders.price \leq priceToCheck].size)$
110:        Assert$((\text{minimum}(buyVolumeNew, sellVolumeNew) < volumeSettled) \vee (\text{minimum}(buyVolumeNew, sellVolumeNew) = volumeSettled \wedge imbalance \leq |buyVolumeNew - sellVolumeNew|))$    ▷ If the next price clears less volume, or clears the same volume with a larger imbalance, the proposed CP is valid
111:        $SettleOrders(CP, buyVolume, sellVolume)$
112:    if $imbalance < 0$ then    ▷ As the auction is offered at $CP$, check if next price increment below clears higher volume OR smaller imbalance
113:        $priceToCheck \leftarrow CP - minTickSize$
114:        $buyVolumeNew \leftarrow (buyVolume + sum(revealedBuyOrders[CP > revealedBuyOrders.price \geq priceToCheck].size))/CP$
115:        $sellVolumeNew \leftarrow sellVolume - sum(revealedSellOrders[CP \geq revealedSellOrders.price > priceToCheck].size)$
116:        Assert$((\text{minimum}(buyVolumeNew, sellVolumeNew) < volumeSettled) \vee (\text{minimum}(buyVolumeNew, sellVolumeNew) = volumeSettled \wedge imbalance \leq |buyVolumeNew - sellVolumeNew|))$
117:        $SettleOrders(CP, buyVolume, sellVolume)$
118:    $protocolContract.transfer(2resBounty, ₿, player))$    ▷ Return deposit, and reward player for submitting valid clearing price
```

Algorithm 9 Resolution: Settle Orders

```
112: function SettleOrders($CP$, $buyVolume$, $sellVolume$)
113:     $buyVolume \leftarrow buyVolume/CP$                    ▷ Convert sell volume to equivalent in $x$
114:     if $buyVolume > sellVolume$ then ▷ pro-rate buy orders at the min price above (or equal to) the clearing price
115:         $price_{pro\text{-}rate} \leftarrow minimum(revealedBuyOrders[revealedBuyOrders.price \geq CP].price)$
116:
         $size_{pro\text{-}rate} \leftarrow sum(revealedBuyOrders[revealedBuyOrders.price = price_{pro\text{-}rate}].size)/CP$
117:         for $order \in revealedBuyOrders[revealedBuyOrders.price = price_{pro\text{-}rate}]$ do
118:             $protocolContract.transfer(order.size \cdot (1 - \frac{buyVolume - sellVolume}{size_{pro\text{-}rate}}), x, order.player)$ ▷ return tokens not going to be exchanged
119:             $order.size \leftarrow order.size \cdot \frac{buyVolume - sellVolume}{size_{pro\text{-}rate}}$
120:     else if $sellVolume > buyVolume$ then ▷ pro-rate sell orders at the max price below (or equal to) the clearing price
121:         $price_{pro\text{-}rate} \leftarrow maximum(revealedSellOrders[revealedSellOrders.price \leq CP].price)$
122:         $size_{pro\text{-}rate} \leftarrow sum(revealedSellOrders[revealedSellOrders.price = price_{pro\text{-}rate}].size)$
123:         for $order \in revealedSellOrders[revealedSellOrders.price = price_{pro\text{-}rate}]$ do
124:             $protocolContract.transfer(order.size \cdot (1 - \frac{sellVolume - buyVolume}{size_{pro\text{-}rate}}), y, order.player)$ ▷ return tokens not going to be exchanged
125:             $order.size \leftarrow order.size \cdot \frac{sellVolume - buyVolume}{size_{pro\text{-}rate}}$

126:     for $order \in revealedBuyOrders, revealedSellOrders$ do               ▷ iterate through orders
127:         if $order \in revealedBuyOrders \wedge (order.price \geq CP \vee order.price = mkt)$ then ▷ execute buy order if bid greater than clearing price
128:             $tokenTradeSize \leftarrow order.size/CP$
129:             $protocolContract.transfer(tokenTradeSize, y, order.player)$
130:         else if $order \in revealedSellOrders \wedge (order.price \leq CP \vee order.price = mkt)$ then ▷ execute sell order if bid greater than clearing price
131:             $tokenTradeSize \leftarrow (order.size)/CP$
132:             $protocolContract.transfer(tokenTradeSize, x, order.player)$
133:         else                                                          ▷ Order not executed
134:             $protocolContract.transfer(order.size, order.tkn, order.player)$

135:     $phase \leftarrow Commit$
136:     $currAucNotional \leftarrow 0$
137:     $revealedBuyOrders, revealedSellOrders \leftarrow []$
138:     $width_{tight} \leftarrow any$
139:     $lastPhaseChange \leftarrow Blockchain.height()$
```

MMs who wish to participate generate a market of the form $\mathit{market} \leftarrow (\mathit{bid}, \mathit{size}_{\mathit{bid}}, \mathit{offer}, \mathit{size}_{\mathit{offer}})$, and submit a transaction directly to the blockchain of the form $\langle \mathit{COMMIT}, \mathit{commit}(\mathit{market})\rangle$. This transaction deposits an escrow of escrow_{MM} to the smart contract. The commitment is then recorded in *MMCommits* (line 30).

User and MM Commit transactions are collected until the Commit phase deadline, *commitDeadline* (line 11), has passed (line 31).

Reveal

A user who committed to trade through a $\langle \mathit{COMMIT}, \mathit{com}, \mathcal{S}, \pi\rangle$ transaction in the Commit phase submits a Reveal transaction directly to the blockchain of the form $\langle \mathit{CLIENT\text{-}REVEAL}, \ \mathit{tkn}, \mathit{size}, \mathit{price}, \mathit{width}, \ \mathcal{S}, \mathcal{R}, \mathit{regID} \ \mathit{regIDNew}\rangle$ (line 34). If the user intends to ren-enter the protocol as a user, *regIDNew* is a commitment to a new serial number and randomness. Otherwise, it is the null value.

This $\langle \mathit{CLIENT\text{-}REVEAL}, *\rangle$ transaction reveals the token being sold, token amount to sell, and requested width of the order. The *price* either reveals a limit price at which the user is selling, that the order is the market order if $\mathit{price} = \mathit{mkt}$, or that the user is withdrawing their escrow if $\mathit{price} = \mathit{withdraw}$. The contract checks:

- $\mathcal{S} \in \mathit{userCommits}$ to verify a commitment corresponding to that serial number has been recorded.

- $\mathit{regID} = \mathit{commit}(\mathcal{S}, \mathcal{R})$ to ensure that user was indeed the same player that generated the *regID* and that the user order is the same as that committed in the Commit phase $\mathit{commit}(\mathit{tkn}, \mathit{size}, \mathit{price}, \mathit{width}) = \mathit{userCommits}[\mathcal{S}].\mathit{com}$.

If the transaction is valid, the order is then added to *revealedBuyOrders* or *revealedSellOrders*, depending on direction. If the token being sold by the user is x, the effective order size for clearing price calculation and trade size allocation is the minimum of *size* and $\mathit{escrow}_{\mathit{user}}/\varepsilon_x$, the maximum token x order size allowable (line 37), with the order recorded in *revealedOrders* (line 38). If the token being sold by the user is y, the order size is the minimum of *size* and $\mathit{escrow}_{\mathit{user}}/(\varepsilon_x \cdot \mathit{offer})$ (line 45), with the order recorded in *revealedOrders* (line 46).

Finally, if *regIDNew* is the null value, the escrow is returned (line 41 or 48), while if it is not, it corresponds to re-entering the protocol as a new user (saving on an additional transaction to re-enter as a user).

A MM who committed to a market with a $\langle \mathit{COMMIT}, \mathit{com}\rangle$ message in the commit phase submits $\langle \mathit{MM\text{-}REVEAL}, \mathit{market} \leftarrow (\mathit{bid}, \mathit{size}_{bid}, \mathit{offer}, \mathit{size}_{offer})\rangle$ (line 88). The contract verifies:

- The market *market* matches the previously commitment from the commit phase ($\mathit{commit}(\ \mathit{market}) = \mathit{MMCommits}[\mathit{MM}].\mathit{com}$) (line 88).

- For the bid, $Q_{not}/\varepsilon_x \leq \mathit{size}_{bid}$, which verifies the MM has provided the minimum notional required by a WSFBA (line 89).

- For the offer, $Q_{not}/(\varepsilon_x \cdot \mathit{offer}) \leq \mathit{size}_{offer}$, which again verifies the MM has provided the minimum notional required by a WSFBA (line 89).

Any MM not revealing a market (line 77) loses their escrow (line 78) and is prevented from participating in the Resolution phase. Otherwise, from the set of all valid revealed markets *revealedMkts*, the tightest market is selected, including a tie-breaking procedure for more than one market with width equal to the tightest width. The tie-breaker used in our implementation of FairTraDEX takes, for a MM *MM* (identified by a unique public key) and submitted market *market*, the market corresponding to the largest value of $\mathit{commit}(\mathit{commit}(\mathit{revealedMkts})\ ||\ \mathit{MM}\ ||\ \mathit{market})$ (lines 57-73). Given the tightest market $\mathit{market} \leftarrow (\mathit{bid}, \mathit{size}_{bid}, \mathit{offer}, \mathit{size}_{offer})$ after tie-breaks, the two implicit limit orders are added to the set of revealed orders *revealedBuyOrders* and *revealedSellOrders*. As was the case with user orders, the effective order size for clearing price calculation and trade size allocation of the bid is the minimum of size_{bid} and $\mathit{escrow}_{MM}/\varepsilon_x$, while the effective offer size is the minimum of size_{offer} and $\mathit{escrow}_{MM}/(\varepsilon_x \cdot \mathit{offer})$.

Reveal transactions are collected until the Reveal deadline, *revealDeadline* (line 11), has passed (line 31).

Resolution

Once the protocol enters the Resolution phase, any player in the system can propose a clearing price by submitting a $\langle \mathit{CP}, *\rangle$ message. Players submitting such a message must deposit a token amount (which we set as *resBounty*, although any significantly large value to prevent invalid calls to the smart contract will do). This deposit, along with a bounty is returned to the player if *CP* is a valid clearing price.

The orders are then settled based on the clearing price *CP* (lines 114-134). If the quantity of x being sold is greater than the quantity being bought, the sell orders at the highest sell price below the clearing price are pro-rated based on the quantity of x being sold. If the quantity of token x being bought is greater than the quantity being sold, the buy orders at the lowest buy price above the clearing price are pro-rated based on the quantity of x being bought. Remaining unexecuted order balances and escrows are returned to the owners.

5.6 Properties of FairTraDEX

We now argue that FairTraDEX possesses all of the necessary properties to instantiate a WSFBA, and discuss the practical implications of these properties. As the Register() and CommitUser() algorithms are constructed analogously to the Mint() and Spend() functions of [81], we can make use of the results as provided therein. These can be translated informally as:

1. Linkability: Consider a player $\mathcal{P}_j$, a set of registrations *RegIDs* to which $\mathcal{P}_j$ does not know the privately committed values, and a valid ZK signature of knowledge π and serial number $\mathcal{S}$ corresponding to some $regID_i \in RegIDs$. $\mathcal{P}_j$ in has no advantage in linking π and $\mathcal{S}$ to the corresponding $regID_i$ over probability $\frac{1}{|RegIDs|} + negl(\kappa)$.

2. Double-spending: Given a set of registrations *RegIDs*, and any number of valid (π, $\mathcal{S}$) pairs corresponding to elements in *RegIDs*, it is computationally infeasible to generate a new serial number $\mathcal{S}'$ and corresponding valid proof of registration $\pi\ '$ in *RegIDs*.

Given that all players in the system are registered as users, by definition of the external market price, the expected trade imbalance implied by their orders is 0. However, in reality, we cannot always expect this level of user participation, with less user registrations typically resulting in a greater advantage for rational players in predicting the implied trade imbalance of committed orders.

To account for this in our analysis, we introduce n_ψ denoting the minimal number of registrations required to guarantee that EEV is $negl(\psi)$. Note that in certain blockchain systems, as the total number of players may be unknown to players within the system, precisely defining n_ψ may not be possible (for example, a player registering for the second time observes a smaller relative increase than a player registering for the first time). In that sense, our analysis demonstrates that the listed desirable properties can be achieved under a sufficient level of registration (n_ψ registrations), but not necessarily that a user can detect whether this level is met in a given auction instance. In practice, a user's decision whether or not to commit to an order in FairTraDEX will be based on heuristics involving the number n_c of observed user registrations, noting that non-negligible EEV may be tolerable if the total expected participation fees are less than *fee*.

Towards proving FairTraDEX forms an SNE for rational users and MMs, we provide a series of Lemmas that we use to prove the main result of the section, Theorem 5.6.8. We first prove that some player in the blockchain protocol runs a Resolution() instance every round.

Lemma 5.6.1. *At least one player runs Resolution() in every round.*

Proof. Consider a Resolution phase where at least one player has not called a CommitUser() or CommitMM() instance in the preceding Commit phase. This player is indifferent to the settlement of orders, and as such the only payoff for that player by running Resolution() correctly is the receipt of $resBounty \in \mathbb{R}^+$.

Consider instead the case where all players in the system called at least one instance of CommitUser() and/or CommitMM() in the preceding Commit phase. In this case, all players have an additional payoff for receipt of the tokens currently locked in the protocol. As at least one of the buyers or sellers of the swap must receive a non-zero amount of tokens, the receipt of which having at worst 0 utility. This, in addition to the receipt of *resBounty* makes the calling of Resolution() positive expectancy for at least one player in the system. □

We now prove a series of Lemmas demonstrating that given a rational user (resp. MM) runs an instance of Register() (resp. CommitMM()), that same player correctly runs CommitUser() and RevealUser() (resp. RevealMM()) in the proceeding phases.

Lemma 5.6.2. *A rational user who correctly runs an instance of Register() also runs correct corresponding instances of CommitUser() and RevealUser().*

Proof. By correctly running Register(), a player deposits $escrow_{user}$. The only way to receive $escrow_{user}$ back is to correctly run a RevealUser() instance, which itself can only be run after having run a CommitUser() instance in the previous phase. By construction, $escrow_{user}$ is greater than any incurrable losses by running CommitUser() and RevealUser(), with maximal losses occurring where the user's order is settled for tokens that have 0 notional (price goes to 0). As the user's initial deposit had notional value strictly less than $escrow_{user}$, so must the incurred loss. The result follows. □

Lemma 5.6.3. *A rational MM who correctly runs an instance of CommitMM() also runs a correct instance of RevealMM() in the proceeding phase.*

Proof. By correctly running CommitMM(), a player deposits $escrow_{MM}$. The only way to receive $escrow_{MM}$ back is to correctly run a RevealMM() instance in the proceeding. By definition, as the MM's bid and offer has notional value greater than or equal to the total notional in the auction, Q_{not}, so must the incurred loss of not revealing a market. Therefore, players running CommitMM() always correctly run RevealMM(). □

Lemma 5.6.4. *Consider an instance of FairTraDEX between x and y. A rational user $\mathcal{P}_i$ with $bal_i(\text{Ƀ}) > escrow_{user}$ runs an instance of Register().*

Proof. Consider such a user $\mathcal{P}_i$ with minimum user utility *fee* to exchange one token for another. To execute the swap, $\mathcal{P}_i$ must first call Register(). Given $\mathcal{P}_i$ calls Register(), we know from Lemma 5.6.2 that $\mathcal{P}_i$ also calls CommitUser() and RevealUser(). We know from Lemma 5.6.1, Resolution() will be run in every round, meaning $\mathcal{P}_i$ either trades or the tokens are returned. Given a trade occurs, $\mathcal{P}_i$ realises the utility of trading given the restrictions of the order generated by $\mathcal{P}_i$ in CommitUser(), which can be chosen to be any value with positive utility (buy below the external market price/sell above the external market price). If no trade occurs, $\mathcal{P}_i$'s realised utility (of 0) does not change. Therefore, $\mathcal{P}_i$ runs Register(). □

The following lemma demonstrates that it is indeed an SNE for a user (resp. MM) to run Register() (resp. CommitMM()).

Lemma 5.6.5. *Consider an instance of FairTraDEX between x and y, and at least 1 previously called instance of Register(). Any rational MM $\mathcal{P}_i$ with $bal_i(\text{Ƀ}) \geq escrow_{MM}$ and $bal_i(x),\ bal_i(y) > 0$ runs an instance of CommitMM().*

Proof. Consider such a player $\mathcal{P}_i$. Given Register() was called by some player $\mathcal{P}_j$, $\mathcal{P}_i$ knows $\mathcal{P}_j$ must call CommitUser() and RevealUser(). Furthermore, $\mathcal{P}_i$ knows the total order size is bounded by Q_{not} (Section 5.4). Given external market price $\varepsilon_{x\rightarrow y}$ and the definition of a MM, there is some market $market \leftarrow (bid@offer)$ at which $\mathcal{P}_i$ observes positive utility to trade with $\mathcal{P}_j$. Therefore, $\mathcal{P}_i$ submits *market* to the auction. We must now ensure $\mathcal{P}_i$ submits the order by calling CommitMM(). By the calculation of order settlement, if $\mathcal{P}_i$ submits *market* through the necessary Register() and CommitUser() calls and $\mathcal{P}_i$ submits a market order with finite requested width, no trade happens. As such, $\mathcal{P}_i$ runs CommitMM(). □

With these lemmas in hand, we have it that rational users and rational MMs correctly execute all algorithms as outlined by FairTraDEX. This can be expressed concisely in the following corollary. In this corollary, and the theorem that follows, we assume users and MMs satisfy the token-balance requirements as described in Lemmas 5.6.4 and 5.6.5 respectively.

Corollary 5.6.6. *Rational users and MMs always follow the FairTraDEX protocol.*

Observation 5.6.7. *It can be seen that FairTraDEX with at least n_ψ previous Register() calls implements a WSFBA when all players follow the protocol. In the Commit phase, CommitUser() specifies a user order which is committed to, while CommitMM() specifies a market which is also committed to. As users commit to these orders and sign this commitment using a NIZKSoK, nothing is revealed about the user's order, as there are are least n_ψ Register() calls. This is equivalent to privately submitting the order.*

CommitMM() and RevealMM() ensure MMs provide the equivalent of at least Q_{not} notional on the bid and offer. Furthermore, as MMs are indistinguishable (see Section 5.3), a MM commitment reveals nothing about the liquidity on the bid or offer[6].

When the Commit phase is finished, no further orders or markets can be submitted for that auction round, and as such, the clearing price is predetermined. During the Reveal phase, RevealUser() and RevealMM() reveal the orders corresponding to CommitUser() and CommitMM() instances from the previous phase, which are settled in the Resolution phase according to clearing price rules which maximise the amount of notional to be traded, as is in a WSFBA.

With these results in hand, we have it that rational users and rational MMs correctly execute all algorithms as outlined by FairTraDEX. We now show that with at least n_ψ Register() calls, the optimal strategy for a user is to submit market orders, while the optimal strategy for a MM with external market price $\varepsilon_{x\to y}$ is to show a market *bid* @ *offer* with $bid \approx \varepsilon_{x\to y} \approx offer$ in the case where there are at least 2 non-cooperative MMs, and of width $width \leq fee$ otherwise.

Theorem 5.6.8. *Consider an instance of FairTraDEX between x and y with external market price $\varepsilon_{x\to y}$ and at least n_ψ previously called instances of Register(). For N non-cooperative MMs, the following strategies form strict Nash equilibria:*

[6]MMs are also allowed to participate as users if privacy is a concern. CommitMM() provides professionals with a functionality to efficiently provide liquidity in a decentralised setting. It is possible to introduce a RegisterMM() function analogous to Register(), allowing MMs to relay markets in ZK. We believe this has negligible benefit for professionals who already have price and market-size hidden through the commitment scheme.

- $N = 1$*: users run Register(), followed by CommitUser() producing market orders of width fee. The MM runs CommitMM() producing a market of width at most fee with reference price equal to* $\varepsilon_{x \rightarrow y}$ *in size* Q_{not}*. Users and MMs then run RevealUser() and RevealMM() respectively.*

- $N \geq 2$*: users run Register(), followed by CommitUser() producing market orders of width greater than 1. MMs run CommitMM() producing markets of width 1 with reference price equal to* $\varepsilon_{x \rightarrow y}$ *in size of at least* Q_{not}*. Users and MMs then run RevealUser() and RevealMM() respectively.*

Proof. From Corollary 5.6.6, we know the running of FairTraDEX is a strictly dominant strategy for users and MMs. Furthermore, from Observation 5.6.7, we have seen that FairTraDEX with at least n_ψ previously called instances of Register() implements a WSFBA. Given FairTraDEX is a WSFBA, the statement follows by applying Theorem 5.4.3. □

Although providing width 1 markets may seem prohibitive for MMs, the unique guarantees of FairTraDEX ensure that no players external to the protocol can extract value from players within the protocol. As player value is being retained within the FairTraDEX protocol, fees can be introduced to compensate MMs. Given the potential value retention of FairTraDEX (see Section 5.8, Table 5.2), these fees can be substantial while still ensuring FairTraDEX provides users with best-in-class liquidity.

Remark 5.6.9. *To minimise expected absolute trade imbalances in a DEX auction, existing protocols, including FairTraDEX, require the hiding/mixing of order-information. Consider how FairTraDEX compares to previous DEXs aimed at ensuring user privacy [22, 40, 51]. In these previous protocols, each order commit reveals the same, and in some cases more information per-order than a Register() call in FairTraDEX. EEV protection guarantees in these previous protocols which require* n_ψ *orders per auction are achieved in FairTraDEX for every order in every auction when* n_ψ *players are registered to participate in the protocol. This is an* n_ψ *factor improvement in EEV protection/block-space requirements per auction.*

More than this, these previous protocols face liveness issues when players are concerned about EEV. The first players entering one of these previous protocols must choose to do so without any guarantees of protection against EEV attacks based on information leaked from order commitment (trade direction, identity, trading patterns, etc.).

5.6.1 Clearing Price Verification

The protocol in Algorithm 8 checks that a given clearing price *CP* clears the highest notional with respect to x. To do this, it checks the imbalance and total notional that would be settled at *CP*. Note, that the statements that follow are true given some volume trades at the proposed clearing price, which is asserted in the protocol (line 99).

If the imbalance is 0 (line 104), it can be seen that this price must maximise the volume traded while minimising the absolute value of the imbalance. Higher prices strictly reduce the buying notional/ strictly increase the selling notional, while lower prices strictly reduce the selling notional/increase the buying notional, which creates an imbalance without increasing the notional traded.

If the imbalance is positive (line 106), this implies there will be buy orders (partially) unfilled at or above *CP*. We can see that the only prices at which more notional can trade must be greater than *CP*. Thus, the contract checks the next price increment above (line 107), and verifies the total notional traded at that price is less than at *CP*, or equal, but with a larger absolute imbalance (line 116). If the notional traded is lower at this higher price, the clearing price is correct as a lower price reduces the value of the selling notional, and increases the value of the buying notional. If the notional traded is the same, but the absolute value of the imbalance is higher, this imbalance must be a sell imbalance by the same reasoning (buying notional decreases, selling notional increases). If the absolute value of the imbalance is higher (although negative), the imbalances at all price points above that price are increasing (buying notional decreases, selling notional increases).

The same holds for sell imbalances at a proposed clearing price (line 112). Checking the price point below (line 113) and ensuring the notional is lower, or that the notional traded is the same but with a large absolute imbalance (line 116) guarantees that the proposed clearing price is valid.

5.7 Notes on FairTraDEX

This section contains several subsections dedicated to addressing some of the questions that arose throughout the development and dissemination of FairTraDEX.

5.7.1 Existence of irrational players and coalitions

When analysing the optimal strategies of rational players in WSFBAs, our results are based on all players being rational and that n_ψ instances of Register() are called. If we consider the presence of irrational players in the system, we can apply the following adjustments:

- **Irrational MM**: In Lemma 5.4.2, it is shown that the optimal strategy for a MM is to show markets centred around the external market price. Any other (irrational) strategy must therefore result in reduced expectancy for the MM, and higher expectancy for users. Therefore, given the presence of irrational MMs, submitting market orders maximises user expectancy (with greater expected utility than in the presence of rational MMs, although with increased variance).

- **Irrational user**: Given the optimal strategy for rational users is to submit market orders, a irrational user may then submit limit orders. This merely reduces the irrational user's chance of trading vs. other users. This would not change the strategy of non-colluding rational MMs, but may have some affect on a monopolistic MM's interpretation of *fee*.

Furthermore, if less than n_ψ instances of Register() are called, users resort to submitting limit orders. This can be seen in the proof of Theorem 5.4.3. In the proof, if users are not sure that a MM will show a market with reference price equal to the external market price, the case when less than n_ψ instances of Register() are called, the optimal strategy for users is to submit limit orders, which only stands to reduce the users' probability of trading. As the number of non-cooperative players in FairTraDEX decreases towards two, the guarantees of FairTraDEX approach those of an AMM. However, as user price and order size remain hidden until the counterparty chooses her strategy, and before the clearing price is fixed (end of the Commit phase), FairTraDEX maintains advantages over AMMs against user-based EEV attacks, such as price/order-size specific front-running and selective participation.

5.7.2 Practical Considerations for FairTraDEX

Practical approaches to ensure equivalent to n_ψ Register() calls are taken in existing anonymity protocols. For example, Tornado Cash [7] rewards players

[7] https://torn.community/t/anonymity-mining-technical-overview/15 Accessed: 20/07/2022

proportionally to the (Tornado Cash equivalent of the) number of Register() calls, as well as the length of time between calling Register() and CommitUser().

Escrow choices

Choosing escrow amounts for users and MMs should reflect the emergent use cases of the protocol. It is possible to create different FairTraDEX instances for the same pair of tokens based on trade size, both for liquidity purposes (MMs will require wider markets for larger-sized orders, but the corresponding increased escrow requirements might prevent smaller users from participating) and auction use-cases (day-trading vs. end-of-day balancing). Furthermore, as the escrow denomination (₿) in our description is different to at least one of the tokens, there needs to be some way to translate the escrow amount into order sizes. This will depend on the environment, but on Ethereum for example, price oracles (existing AMMs, Chainlink[8], etc.) can be used. It is also possible to use previous clearing prices from within the FairTraDEX ecosystem, although a self-referential oracle must be implemented carefully as there may be game-theoretic implications. If a satisfactory price oracle exists, deposits can be made in the respective tokens of the swap, further reducing the capital requirements for players and encouraging adoption.

Incentive compatibility given transaction fees

In Section 5.5, we mention that our strict Nash equilibria are dependent on the utility gained by users and MMs being greater than the cost for participation. The choice of smart-contract enabled blockchain on which to deploy will dictate the barrier of entry for users and MMs alike. Like existing attempts to implement blockchain-based FBAs [51, 40], we have an amortised number of transactions per player of two. A naive comparison to AMMs, where this is reduced to 1 for users, and 0 for MMs, certainly has less direct costs than FairTraDEX. However, when factors like impermanent loss, slippage[9], front-running, and EEV attacks in general, the value being extracted from DEXs incurs a significant indirect cost for users. Immediately, we can increase the expected cost of using AMMs by the slippage required by AMMs (set to 0.5% as of writing in Uniswap V3, but for larger orders this must increase by the nature of AMMs). We can increase this further by the probability orders are not executed (where prices move more than the slippage, potentially due to front-running)

[8] https://chain.link/

[9] https://docs.uniswap.org/protocol/concepts/V3-overview/swaps

but are added to the blockchain. As such, the indirect costs are substantial, are increasing in order-size increases and proportionally to improvements in user trading ability as strategies can be replicated/front-run. A thorough comparison of the monetary costs of FairTraDEX vs. AMMs over various order-sizes, and trading scenarios makes for interesting future work as FairTraDEX begins to be deployed and tested in the wild.

It can be seen from the proof of Theorem 5.4.3 that the user strategies identified are strong incentive compatible in expectation as the MM always shows markets of width less than or equal to *fee*. However, the MM strategy of showing width 1 markets is not strong incentive compatible. In addition to the fees described in the protocol, an additional fee can be applied within the protocol itself to incentivize the participation of MMs. This can be a function of MM participation/market widths. As with all additional fees/rewards, the game-theoretic implications of such an incentivization scheme must be considered.

In our encoding of FairTraDEX, we do not explicitly introduce a cost for users and MMs in Commit/Reveal phases to reward the submission of the clearing price. In reality, the result in Lemma 5.6.1 holds without the introduction of an explicit reward, as there all participating users and MMs will have positive expectancy to receive tokens through correct order resolution. The use of an explicit reward is for illustrative purposes, and to avoid complications regarding transaction fees for running the clearing price checks. The costs of running the Resolution contracts must be ensured to be less than the utility gained by at least one player in the blockchain protocol for calling the contract.

5.8 Cost-Benefit Analysis of FairTraDEX

The aim of this section is to demonstrate the contributory significance of FairTraDEX vs. current state-of-the-art protocols as introduced in Section 5.2. Our results are based on the Solidity implementation of the protocol provided in [54]. In Table 3.1 we include an overview of the gas costs for running FairTraDEX compared to the previous blockchain-based attempts to implement batch auctions of [51, 40], with numbers taken from the respective papers. These are the fixed costs for including and executing the transactions on a blockchain. It can be seen that FairTraDEX has a slightly greater upfront gas cost for users, but a lesser cost for MMs. This is directly related to the added costs of correctly using ZK tools to hide users identity and order information until all orders have been committed. Compared to the variable costs of revealing this information, we see these costs as acceptable.

	FairTraDEX [10]	Uniswap	[40]	[51]
Register	112,800	-	87,000	-
Commit User	344,500	-	52,000	276,150
Commit MM (per order)	24,300	-	52,000	276,150
Reveal (per order)	172,000	190,000	171,000	48,750
Settle (per order) [11]	45,500	-	122,500	54,000
Total User	674,800	190,000	432,500	378,900
Total MM	266,100	-	397,500	649,050
Total User (USDC)	7.27	2.05	4.66	4.08
Total MM (USDC)	2.87	-	4.09	7

Table 5.1: Comparison of gas costs in batch-auction implementations. [10] Costs provided for FairTraDEX are amortised over 128 user orders and 8 markets. [11] We add an estimated cost for token transfer from smart contract to player of 40,000 to the figures provided in [51] to standardise the costs therein with those of FairTraDEX and [40].

To demonstrate the benefits of FairTraDEX, Table 5.2 compares specific swaps that allow for EEV attacks in existing state-of-the-art protocols. We perform our analysis on ETH/USDC swaps, as this is the highest volume pool on Uniswap, which at time of writing had pool sizes of 120k ETH and 185M USDC, an indicative EMP of 1 ETH equal to 1,540 USDC [102]. Furthermore, we use a gas cost of 7 gwei [46]. Consider 3 buy ETH orders of 10k, 500k and 10M USDC from 2 different players who are known to need to trade at any price. $\mathcal{P}_1$ has large quantities of both ETH and USDC, and buys or sells ETH pseudo-randomly, while $\mathcal{P}_2$ only owns USDC/only buys ETH. We take the estimated impact for each order to be 0, 0.15% and 1% respectively, numbers taken from the Uniswap V3 API [102] (these are more realistic impacts than those implied by the constant product impact [13] of 0, 0.54% and 11.1% respectively). Although this is a simplification of order impact, true impact is likely some multiple/factor of this impact. Protocol fees incentivizing MMs to provide liquidity are omitted as they are not considered in the provided academic protocols. After gas costs, this fee should be approximately equal for all protocols (the Uniswap fee for this pool is 0.3%).

When $\mathcal{P}_1$ submits an order in FairTraDEX or [51], no information is gained about the direction of the trade. However, in [40], direction is revealed. As such, any blockchain participant can front run that impact on all other markets, and thus the EMP for any MM responding to the order will be the impacted EMP. When $\mathcal{P}_2$ submits an order in either of [40, 51] the direction is known, and the EMP is impacted in the same way as for $\mathcal{P}_1$. Crucially, this impact takes place

	FairTraDEX	Uniswap	[40]	[51]
$\mathcal{P}_1$-10,000	7	52	5	4
$\mathcal{P}_2$-10,000	7	52	5	4
$\mathcal{P}_1$-500,000	7	3002	755	4
$\mathcal{P}_2$-500,000	7	3002	755	754
$\mathcal{P}_1$-10,000,000	7	150,002	100,005	4
$\mathcal{P}_2$-10,000,000	7	150,002	100,005	100,004

Table 5.2: Comparison of expected execution costs in USDC of batch-auction implementations, including the costs of Table 5.1.

before any player interacts with $\mathcal{P}_2$, giving $\mathcal{P}_2$ a worse price. Using estimated price impacts of 0, 0.15% and 1%, Table 5.2 demonstrates the costs of executing these swaps, excluding transaction fees, in these protocols, and Uniswap. For Uniswap, we must also add the recommended slippage, an additional 0.5% of the order size, as it is always in a block producers interest to give Uniswap players worst execution. It can be seen that these costs become increasingly more significant as order size increases, dominating the differences in gas costs of Table 3.1.

Although Table 5.2 can be seen as simplifying how orders are handled, it demonstrates two crucial motivators for our work. Firstly, any information revealed about users before a trade is agreed can, is and will continue to be used against users. Furthermore, this cost is not necessarily paid to the MM. As orders are committed in public, any blockchain participant can use the committed information to front run the impact on the EMP before the MM or users have an opportunity to trade, extracting money from the DEX protocol. Secondly, as the effects of these value-extraction techniques increase super-linearly in order-size, a protocol with the value-extraction guarantees of FairTraDEX is needed to allow typically large users to utilise the benefits of DEXs, and blockchain protocols in as a whole, at a fixed cost, as demonstrated in Table 3.1, without incurring the prohibitive execution costs of previous solutions, as demonstrated in Table 5.2.

5.9 Conclusion

We provide FairTraDEX, a blockchain-based DEX protocol based on WSFBAs in which we formally prove the strategies of rational participants have strict Nash equilibria in which all trades occur at the external market price plus or minus bounded upfront costs (specified market widths) which approach 0 in the presence of non-cooperative MMs. This is an attractive alternative to existing mainstream protocols such as AMMs where rational players effectively and systematically prevent such an equilibrium from happening. Compared to previous blockchain-based attempts to implement EEV-proof DEXs, FairTraDEX is the first to practically allow for indistinguishable user-order submissions by decoupling order submission from escrow deposit and order revelation. The FairTraDEX benefits formalised in Section 5.6, summarised in Remark 5.6.9, and demonstrated in Section 5.8 provide important improvements on previous protocols regarding EEV protection, setting a new standard for EEV protection in DEXs.

As stated in the comparisons of Section 5.8, protocol fees are omitted for all protocols. Given the total retention of value within the FairTraDEX protocol (no extractable value), fees in line with the utility gained by users for exchanging their tokens can be charged to incentivize the long-term participation of MMs in FairTraDEX. These fees should reflect the need to incentivize MMs while retaining the unique user-side benefit of trading at the external market price in expectation, which is proven to occur in FairTraDEX. Analysis of these fees makes for interesting future work.

Chapter 6

Diamond

This chapter is based on the paper *Diamonds are Forever, Loss-Versus-Rebalancing is Not* [78].

6.1 Introduction

CFMMs such as Uniswap [102] have emerged as the dominate class of AMM protocols. CFMMs offer several key advantages for decentralized liquidity provision. They are efficient computationally, have minimal storage needs, matching computations can be done quickly, and liquidity providers can be passive. Thus, CFMMs are uniquely suited to the severely computation- and storage-constrained environment of blockchains.

Unfortunately, the benefits of CFMMs are not without significant costs. One of these costs is definitively formalized in [82] as *loss-versus-rebalancing* (LVR). It is proved that as the underlying price of a swap moves around in real-time, the discrete-time progression of AMMs leave arbitrage opportunities against the AMM. In centralized finance, market makers typically adjust to new price information before trading. This comes at a considerable cost to AMMs (for CFMMs, [82] derives the cost to be quadratic in realized moves), with similar costs for AMMs derived quantitatively in [90, 35].

These costs are being realized by liquidity providers in current AMM protocols. Furthermore, toxic order flow, of which LVR is a prime example, is consistently profiting against AMM liquidity providers (Figure 6.1). All of these factors point towards unsatisfactory protocol design, and a dire need for an

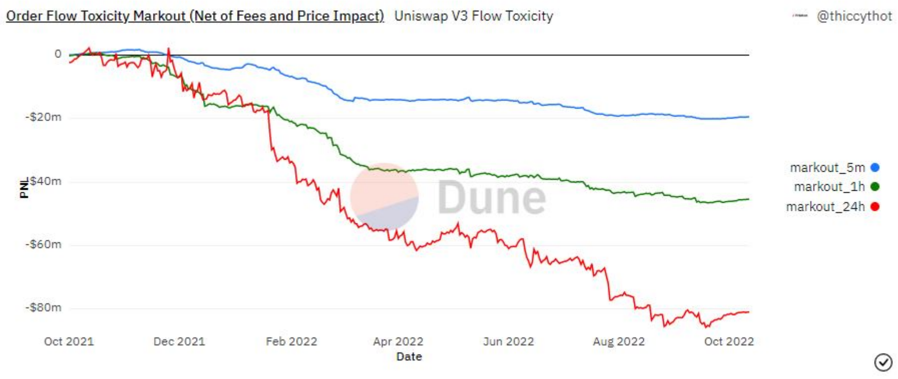

Figure 6.1: Toxicity of Uniswap V3 Order Flow [100]. This graph aggregates the PnL of all trades on the Uniswap V3 WETH/USDC pool, measuring PnL of each order after 5 minutes, 1 hour, and 1 day. This demonstrates the current losses being suffered by AMM pools are significant, consistent, and unsustainable. As LVR is significant and consistent, a large part of these losses can be prevented by minimizing LVR.

LVR-resistant automated market maker. In this chapter, we provide Diamond, an AMM protocol which formally protects against LVR.

6.1.1 Our Contribution

We present Diamond, an AMM protocol which isolates the LVR being captured from a Diamond liquidity pool by arbitrageurs, and forces these arbitrageurs to repay some percentage of this LVR to the pool. As in typical CFMMs, Diamond pools are defined with respect to two tokens x and y. At any given time, the pool has reserves of R_x and R_y of both tokens, and some pool pricing function[1] $PPF(R_x, R_y)$. We demonstrate our results using the well-studied Uniswap V2 pricing function of $PPF(R_x, R_y) = \frac{R_x}{R_y}$.

We introduce the concept of its corresponding CFMM pool for each Diamond pool. For a Diamond pool with token reserves (R_x, R_y) and pricing function $PPF(R_x, R_y) = \frac{R_x}{R_y}$, the corresponding CFMM pool is the Uniswap V2 pool with reserves (R_x, R_y). If an arbitrageur tries to move the price of the corresponding CFMM pool adding Υ_x tokens and removing Υ_y, the same price is achieved in the Diamond pool by adding $(1-\beta)\Upsilon_x$ tokens for some $\beta > 0$, with β the *LVR*

[1]See Equation 6.1 for a full description of pool pricing functions as used in this chapter

rebate parameter. The arbitrageur receives $(1-\beta)\Upsilon_y$. In our framework, it can be seen that $PPF(R_x + (1-\beta)\Upsilon_x, R_y - (1-\beta)\Upsilon_y) < PPF(R_x + \Upsilon_x, R_y - \Upsilon_y)$, which also holds in our example, as $\frac{R_x+(1-\beta)\Upsilon_x}{R_y-(1-\beta)\Upsilon_y} < \frac{R_x+\Upsilon_x}{R_y-\Upsilon_y}$. A further η_y tokens are removed from the Diamond pool to move the reserves to the same price as the corresponding CFMM pool, with these tokens added to a *vault.*

Half of the tokens in the vault are then periodically converted into the other token (at any time, all tokens in the vault are of the same denomination) in one of the following ways:

1. An auction amongst arbitrageurs.

2. Converted every block by the arbitrageur at the final pool price. If the arbitrageur must buy $\frac{\eta}{2}$ tokens to convert the vault, the arbitrageur must simultaneously sell $\frac{\eta}{2}$ futures which replicate the price of the token to the pool. These futures are then settled periodically, either by

   (a) Auctioning the $\frac{\eta}{2}$ tokens corresponding to the futures to the arbitrageurs, with the protocol paying/collecting the difference.

   (b) The use of a decentralized price oracle. In this chapter, we consider the use of the settlement price of an on-chain frequent batch auction, such as that of Chapter 5, which is proven to settle at the external market price in expectancy.

Importantly, these auctions are not required for protocol liveness, and can be arbitrarily slow to settle. We prove that all of these conversion processes have 0 expectancy for the arbitrageur or Diamond pool, and prove that the LVR of a Diamond pool is $(1-\beta)$ of the corresponding CFMM pool. Our implementation of Diamond isolates arbitrageurs from normal users, using the fact that only arbitrageurs compete to capture LVR. This ensures the protections of Diamond can be provided in practice while providing at least the same trading experience for normal users. Non-arbitrageur orders in a Diamond pool are performed identically to orders in the corresponding CFMM pool. Although this means orders remain exposed to the front-running, back-running and sandwich attacks of corresponding CFMMs, the LVR retention of Diamond pools should result in improved liquidity and reduced fees for users.

We discuss practical considerations for implementing Diamond, including decreasing the LVR rebate parameter, potentially to 0, during periods of protocol inactivity until transactions are processed, after which the parameter should be reset. This ensures the protocol continues to process user transactions, which becomes necessary when arbitrageurs are not actively extracting LVR. If arbitrageurs are not arbitraging the pool for even small LVR rebate parameters, it makes sense to allow transactions to be processed as if no LVR was possible. In this case, Diamond pools perform identically to corresponding CFMM pools. However, if/when arbitrageurs begin to compete for LVR, we expect LVR rebate parameters to remain high.

We present a series of experiments in Section 6.7 which isolate the benefits of Diamond. We compare a Diamond pool to its corresponding Uniswap V2 pool, as well as the strategy of holding the starting reserves of both tokens, demonstrating the power of Diamond. We isolate the effects of price volatility, LVR rebate parameter, pool fees, and pool duration on a Diamond pool. Our experiments provide convincing evidence that the relative value of a Diamond pool to its corresponding Uniswap V2 pool is increasing in each of these variables. These experiments further evidence the limitations of current CFMMs, and the potential of Diamond.

6.1.2 Organization of the Chapter

Section 6.2 analyzes previous work related to LVR in AMMs. Section 6.3 outlines the terminology used in the chapter, including chapter-specific player definitions needed to formally reason about Diamond. Section 6.4 introduces the Diamond protocol. Section 6.5 proves the properties of Diamond. Section 6.6 describes how to implement the Diamond protocol, and practical considerations which should be made. Section 6.7 provides an analysis Diamond over multiple scenarios and parameters, including a comparison to various reference strategies. We conclude in Section 6.8.

6.2 Related Work

There are many papers on the theory and design of AMMs, with some of the most important including [5, 4, 82, 20, 21]. The only peer-reviewed AMM design claiming protection against LVR [67] is based on live price oracles. The AMM must receive the price of a swap before users can interact with the pool. Such sub-block time price data requires centralized sources which are prone to manipulation, or require the active participation of AMM representatives, a contradiction of the passive nature of AMMs and their liquidity providers. We see this as an unsatisfactory dependency for DeFi protocols.

Attempts to provide LVR protection without explicit use of oracles either use predictive fees for all players [47] and/or reduce liquidity for all players through more complex constant functions [27]. Charging all users higher fees to compensate for arbitrageur profits reduces the utility of the protocol for genuine users, as does a generalized liquidity reduction. In Diamond, we only reduce liquidity for arbitrageurs (which can also be seen as an increased arbitrageur-specific fee), providing at least the same user experience for typical users as existing AMMs without LVR protection.

A recent proposed solution to LVR published in a blog-post [60] termed *MEV-capturing AMMs* (McAMMs) considers auctioning off the first transaction/series of transaction in an AMM among arbitrageurs, with auction revenue paid in some form to the protocol. Two important benefits of Diamond compared to the proposed McAMMs are the capturing of realized LVR in Diamond as opposed to predicted LVR in McAMMs, and decentralized access to Diamond compared to a single point of failure in McAMMs.

In McAMMs, bidders are required to predict upcoming movements in the AMM. Bidders with large orders to execute over the period (e.g. private price information, private order flow, etc.) have informational advantages over other bidders. Knowing the difference between expected LVR excluding this private information vs. true expected LVR allows the bidder to inflict more LVR on the AMM than is paid for. As this results in better execution for the winner's orders, this may result in more private order flow, which exacerbates this effect. Diamond extracts a constant percentage of the true LVR, regardless of private information. McAMMs also centralize (first) access control to the winning bidder. If this bidder fails to respond or is censored, user access to the protocol is prohibited/more expensive. Diamond is fully decentralized, incentive compatible and can be programmed to effectively remove LVR in expectancy. Future McAMM design improvements based on sub-block time auctions are upper-bounded by the current protection provided by Diamond.

6.3 Preliminaries

This section introduces the key terminology and definitions needed to understand LVR, the Diamond protocol, and the proceeding analysis. In this work we are concerned with a single swap between token x and token y. We use x and y subscripts when referring to quantities of the respective tokens. The external market price of a swap is denoted by a lowercase p, while pool prices/ price functions are denoted using an uppercase P, with the price of a swap quoted as the quantity of token x per token y.

6.3.1 Constant Function Market Makers

A CFMM is characterized by *reserves* $(R_x, R_y) \in \mathbb{R}^2_+$ which describes the total amount of each token in the pool. The price of the pool is given by *pool price function* $PPF : \mathbb{R}^2_+ \to \mathbb{R}$ taking as input pool reserves (R_x, R_y). PPF has the following properties:

$$\begin{aligned} &\text{(a) } PPF \text{ is everywhere differentiable, with } \frac{\partial PPF}{\partial R_x} > 0,\ \frac{\partial PPF}{\partial R_y} < 0. \\ &\text{(b) } \lim_{R_x \to 0} PPF = 0,\ \lim_{R_x \to \infty} PPF = \infty,\ \lim_{R_y \to 0} PPF = \infty,\ \lim_{R_y \to \infty} PPF = 0. \\ &\text{(c) If } PPF(R_x, R_y) = p, \text{ then } PPF(R_x + cp, R_y + c) = p,\ \forall c > 0. \end{aligned} \tag{6.1}$$

These are typical properties of price functions. Property (a) states the price of y is increasing in the number of x tokens in the pool and decreasing in the number of y tokens. Property (b) can be interpreted as any pool price value is reachable for a fixed R_x, by changing the reserves of R_y, and vice versa. Property (c) states that adding reserves to a pool in a ratio corresponding to the current price of the pool does not change the price of the pool. These properties trivially hold for the Uniswap V2 price function of $\frac{R_x}{R_y}$, and importantly allow us to generalize our results to a wider class of CFMMs.

For a CFMM, the *feasible set of reserves* C is described by:

$$C = \{(R_x, R_y) \in \mathbb{R}^2_+ : PIF(R_x, R_y) = k\} \tag{6.2}$$

where $PIF : \mathbb{R}^2_+ \to \mathbb{R}$ is the pool invariant and $k \in \mathbb{R}$ is a constant. The pool is defined by a smart contract which allows any player to move the pool reserves from the current reserves $(R_{x,0}, R_{y,0}) \in C$ to any other reserves $(R_{x,1}, R_{y,1}) \in C$ if and only if the player provides the difference $(R_{x,1} - R_{x,0}, R_{y,1} - R_{y,0})$.

Whenever an arbitrageur interacts with the pool, say at time t with reserves $(R_{x,t}, R_{y,t})$, we assume as in [82] that the arbitrageur maximizes their profits by exploiting the difference between $PPF(R_{x,t}, R_{y,t})$ and the external market price at time t, denoted ε_t. To reason about this movement, we consider a *pool value function* $V : \mathbb{R}_+ \to \mathbb{R}$ defined by the optimization problem:

$$V(\varepsilon_t) = \min_{(R_x, R_y) \in \mathbb{R}^2_+} \varepsilon_t R_y + R_x, \text{ such that } PIF(R_x, R_y) = k \tag{6.3}$$

Given an arbitrageur interacts with the pool with external market price ε_t, the arbitrageur moves the pool reserves to the (R_x, R_y) satisfying $V(\varepsilon_t)$.

6.3.2 Loss-Versus-Rebalancing

LVR, and its prevention in AMMs is the primary focus of this chapter. The formalization of LVR [82] has illuminated one of the main costs of providing liquidity in CFMMs. The authors of [82] provide various synonyms to conceptualize LVR. In this chapter, we use the opportunity cost of arbitraging the pool against the external market price of the swap, which is proven to be equivalent to LVR in Corollary 1 of [82]. The LVR between two blocks B_t and B_{t+1} where the reserves of the AMM at the end of B_t are $(R_{x,t}, R_{y,t})$ and the external market price when creating block B_{t+1} is ε_{t+1} is:

$$R_{x,t} + R_{y,t}\varepsilon_{t+1} - V(\varepsilon_{t+1}) = (R_{x,t} - R_{x,t+1}) + (R_{y,t} - R_{y,t+1})\varepsilon_{t+1}. \tag{6.4}$$

As this is the amount being lost to arbitrageurs by the AMM, this is the quantity that needs to be minimized in order to provide LVR protection. In Diamond, this minimization is achieved.

6.3.3 Auctions

To reason about the incentive compatibility of parts of our protocol, we outline some basic auction theory results.

First-price-sealed-bid-auction: There is a finite set of players $\mathcal{I}$ and a single object for sale. Each bidder $i \in \mathcal{I}$ assigns a value of X_i to the object. Each X_i is a random variable that is independent and identically distributed on some interval $[0, V_{max}]$. The bidders know its realization x_i of X_i. We will assume that bidders are risk neutral, that they seek to maximize their expected payoff. Per auction, each player submit a bid b_i to the auctioneer. The player with the highest bid gets the object and pays the amount bid. In case of tie, the winner of the auction is chosen randomly. Therefore, the utility of a player $i \in \mathcal{I}$ is

$$u_i(b_i, b_{-i}) = \begin{cases} \frac{x_i - b_i}{m}, & \text{if } b_i = \max_i\{b_i\}, \\ 0, & \text{otherwise} \end{cases}$$

where $m = |\mathrm{argmax}_i\{b_i\}|$. In our protocol, we have an amount of tokens z that will be auctioned. This object can be exchanged by all players at the external market price ε. In this scenario, we have the following lemma.

Lemma 6.3.1. *Let $\mathcal{I}$ be a set of players that can exchange at some market any amount of tokens x or y at the external market price ε. If an amount z of token y is auctioned in a first-price auction, then the maximum bid of any Nash equilibrium is at least $z\varepsilon$.*

Proof. By construction, we have that the support of X_i is lower bounded by $z\varepsilon$. Therefore, in a second-price auction, in equilibrium, each player will bid at least, $z\varepsilon$. Using the revenue equivalence theorem [66], we deduce that the revenue of the seller is at least $z\varepsilon$ obtaining the result. □

In one variation of Diamond, it possible to use a periodically updated price oracle to ensure the incentive compatibility of the protocol. To instantiate a game-theoretically secure decentralized price oracles, we can use the settlement price of a decentralized frequent batch auction (see Chapter 5). Frequent batch auctions have been proven to settle in expectancy at the external market price when the auction is run [33]. Such guarantees were originally intended to benefit the users of the frequent batch auction. However, given the settlement price of the auction has expectancy equal to the external market price at a specific time, we can use this as a practical and secure price oracle for settling derivatives depending on this price, without the need for centralized alternatives like Chainlink [36].

6.4 Diamond

This section introduces the Diamond protocol. When the core protocol of Section 6.4.2 is run, some amount of tokens are removed from the pool and placed in a *vault*. These vault tokens are eventually re-added to the pool through a conversion protocol. Sections 6.4.3 and 6.4.4 detail two conversion protocols which can be run in conjunction with the core Diamond protocol. Which conversion protocol to use depends on the priorities of the protocol users, with a comparison of their trade-offs provided in Section 6.7. These trade-offs can be summarized as follows:

- The process of Section 6.4.3 ensures the available liquidity is strictly increasing in expectancy every block, and can be used in conjunction with a decentralized price oracle to ensure the only required participation of a arbitrageurs is in arbitraging the pool (see process 2 in Section 6.4.3).

- The process in Section 6.4.4 incurs less variance in the total value of tokens owned by the pool (see Figure 6.2), and involves a more straightforward use of an auction.

Section 6.5 formalizes the properties of Diamond, culminating in Theorem 6.5.4, which states that Diamond can be parameterized to reduce LVR arbitrarily close to 0. It is important to note that Diamond is not a CFMM, but the rules for adjusting pool reserves are dependent on a CFMM.

6.4.1 Model Assumptions

We outline here the assumptions used when reasoning about Diamond. Inkeeping with the seminal analysis of [82], we borrow a subset of the assumptions therein, providing here a somewhat more generalized model.

1. External market prices follow a martingale process.

2. The risk-free rate is 0.

3. There exists a population of arbitrageurs able to frictionlessly trade at the external market price, who continuously monitor and periodically interact with AMM pools.

4. An optimal solution (R_x^*, R_y^*) to Equation 6.3 exists for every external market price $\varepsilon \geq 0$.

The use of futures contracts in one version of the Diamond protocol makes the risk-free rate an important consideration for implementations of Diamond. If the risk free rate is not 0, the profit or loss related to owning token futures vs. physical tokens must be considered. Analysis of a non-zero risk-free rate is beyond the scope of the thesis.

6.4.2 Core Protocol

We now describe the core Diamond, which is run by all Diamond variations. A Diamond pool Φ is described by reserves (R_x, R_y), a pool pricing function *PPF*(), a pool invariant function *PIF*(), an *LVR-rebate parameter* $\beta \in (0,1)$, and *conversion frequency* $\tau \in \mathbb{N}$.

We define the *corresponding CFMM pool* of Φ, denoted *CFMM*(Φ), as the CFMM pool with reserves (R_x, R_y) whose feasible set is described by pool invariant function *PIF*() and pool constant $k = \mathit{PIF}(R_x, R_y)$. Conversely, Φ is the *corresponding Diamond pool* of *CFMM*(Φ). It is important to note that *CFMM*(Φ) changes every time the Φ pool reserves change. The protocol progresses in blocks, with one reserve update per block.

Consider pool reserves $(R_{x,0}, R_{y,0})$ in Φ and a player wishing to move the price of Φ to $p_1 \neq \frac{R_{x,0}}{R_{y,0}}$. For a player wishing to move the price of *CFMM*(Φ) to p_1 from starting reserves $(R_{x,0}, R_{y,0})$, let this require $\Upsilon_y > 0$ tokens to be added to *CFMM*(Φ), and $\Upsilon_x > 0$ tokens to be removed from *CFMM*(Φ). The same price in Φ is achieved by the following process[2]:

[2]If $\Upsilon_y > 0$ tokens are to be removed from *CFMM*(Φ) with $\Upsilon_x > 0$ tokens to be added in order to achieve p_1, then $(1-\beta)\Upsilon_y$ tokens are removed from Φ and $(1-\beta)\Upsilon_x$ tokens

1. Adding $(1-\beta)\Upsilon_y$ tokens to Φ and removing $(1-\beta)\Upsilon_x$ tokens.
2. Removing $\upsilon_x > 0$ tokens such that:

$$PPF(R_{x,0} - (1-\beta)\Upsilon_x - \upsilon_x, R_{y,0} + (1-\beta)\Upsilon_y) = p_1.^3 \tag{6.5}$$

These υ_x tokens are added to the *vault* of Φ.

After this process, let there be $(\upsilon_x, \upsilon_y) \in \mathbb{R}^2_+$ tokens in the vault of Φ. If $\upsilon_y \varepsilon_1 > \upsilon_x$, add $(\upsilon_x, \frac{\upsilon_x}{\varepsilon_1})$ tokens into Φ from the vault. Otherwise, add $(\upsilon_y \varepsilon_1, \upsilon_y)$ tokens into Φ from the vault. This is a *vault rebalance.*

Every τ blocks, after the vault rebalance, the protocol converts half of the tokens still in the vault of Φ (there can only be one token type in the vault after a vault rebalance) into the other token in Φ according to one of either conversion process 1 (Section 6.4.3) or 2 (Section 6.4.4). After the conversion process, all tokens still in the vault of Φ are added into the Φ pool.

6.4.3 Per-block Conversion vs. Future Contracts

After every arbitrage, the arbitrageur converts η equal to half of the total tokens in the vault at the pool price p_c, equivalent to buying η tokens for p_c. Simultaneously, the arbitrageur sells to the pool η future contracts in the same token denomination at price p_c. Given the pool buys η future contracts at conversion price p_c, and the futures settle at price ε_T, the protocol wins $\eta(\varepsilon_T - p_c)$.

These future contracts are settled every τ blocks, with the net profit or loss being paid in both tokens, such that for a protocol settlement profit of PnL measured in token x and pool price p_T, the arbitrageur pays (s_x, s_y) with $PnL = s_x + s_y p_T$ and $s_x = s_y p_T$. These contracts can be settled in one of the following (non-exhaustive) ways to settle futures:

1. Every τ blocks, an auction takes place to buy the offered tokens from the players who converted the pool at the prices at which the conversions took place. For a particular offer, a positive bid implies the converter lost/the pool won to the futures. In this case the converter gives the tokens to the auction winner, while the pool receives the winning auction bid. A negative bid implies the converter won/the pool lost to the futures. In this case, the converter must also give the tokens to the auction winner, while the pool must pay the absolute value of the winning bid to the auction winner.

are added to Φ, with a further $\upsilon_y > 0$ removed from Φ and added to the vault such that $PPF(R_{x,0} + (1-\beta)\Upsilon_x, R_{y,0} - (1-\beta)\Upsilon_y - \upsilon_y) = p_1$.

[3] Achievable as a result of properties(a) and (b) of Equation 6.1.

2. Every τ blocks, a blockchain-based frequent batch auction takes place in the swap corresponding to the pool swap. The settlement price of the frequent batch auction is used as the price at which to settle the futures.

6.4.4 Periodic Conversion Auction

Every τ blocks, η equal to half of the tokens in the vault are auctioned to all players in the system, with bids denominated in the other pool token. For winning bid b in token x (or token y), the resultant vault quantities described by $(s_x = b, s_y = \eta)$ (or $(s_x = \eta, s_y = b)$) are added to the pool reserves. In this case, unlike in Section 6.4.3, there are no restrictions placed on $\frac{s_x}{s_y}$. As such, they may be in a different ratio than the pool reserves.

6.5 Diamond Properties

This section outlines the key properties of Diamond. We first prove that both conversion process have at least 0 expectancy for the protocol.

Lemma 6.5.1. *Converting the vault every block vs. future contracts has expectancy of at least 0 for a Diamond pool.*

Proof. Consider a conversion of η tokens which takes place at time 0. Let the conversion be done at some price p_c, while the external market price is ε_0. WLOG let the protocol be selling η y tokens in the conversion, and as such, buying η y token futures at price p_c. The token sells have expectancy $\eta(p_c - \varepsilon_0)$. For the strategy to have at least 0 expectancy, we need the futures settlement to have expectancy of at least $\eta(\varepsilon_0 - p_c)$. In Section 6.4.3, two versions of this strategy were outlined. We consider both here. In both subproofs, we use the assumption that the risk-free rate is 0, which coupled with our martingale assumption for ε means the external market price at time t is such that $\mathbb{E}(\varepsilon_t) = \varepsilon_0$. We now consider the two options for settling futures outlined in Section 6.4.3

Option 1: Settle futures by auctioning tokens at the original converted price. The arbitrageur who converted tokens for the pool at price p_c must auction off the tokens at price p_c. Let the auction happen at time t, with external market price at that time of ε_t. Notice that what is actually being sold is the right, and obligation, to buy η tokens at price p_c. This has value $\eta(\varepsilon_t - p_c)$, which can be negative. As negative bids are paid to the auction winner by the protocol, and positive bids are paid to the protocol, we are able to apply Lemma 6.3.1. As such, the winning bid is at least $\eta(\varepsilon_t - p_c)$, which has expectancy of at least

$$\mathbb{E}(\eta(\varepsilon_t - p_c)) = \eta(\mathbb{E}(\varepsilon_t) - p_c) = \eta(\varepsilon_0 - p_c). \tag{6.6}$$

Thus the expectancy of owning the future for the protocol is at least $\eta(\varepsilon_0 - p_c)$, as required.

Option 2: Settle futures using frequent batch auction settlement price. For a swap with external market price ε_t at time t, a batch auction in this swap settles at ε_t in expectancy (Theorem 5.6.8). Thus the futures owned by the protocol have expectancy

$$\mathbb{E}(\eta(\varepsilon_t - p_c)) = \eta(\mathbb{E}(\varepsilon_t) - p_c) = \eta(\varepsilon_0 - p_c). \tag{6.7}$$

□

Lemma 6.5.2. *A periodic conversion auction has expectancy of at least 0 for a Diamond pool.*

Proof. Consider a Diamond pool Φ with vault containing 2η tokens. WLOG let these be of token y. Therefore the pool must sell η tokens at the external market price to balance the vault. Let the conversion auction accept bids at time t, at which point the external market price is ε_t. For the auction to have expectancy of at least 0, we require the winning bid to be at least $\eta\varepsilon_t$. The result follows from Lemma 6.3.1. □

Corollary 6.5.3. *Conversion has expectancy of at least 0 for a Diamond pool.*

With these results in hand, we now prove the main result of the chapter. That is, the LVR of a Diamond pool is $(1 - \beta)$ of the corresponding CFMM pool.

Theorem 6.5.4. *For a CFMM pool $CFMM(\Phi)$ with LVR of $L > 0$, the LVR of Φ, the corresponding pool in Diamond, has expectancy of at most $(1 - \beta)L$.*

Proof. To see this, we first know that for $CFMM(\Phi)$ at time t with reserves $(R_{x,t}, R_{y,t})$, LVR corresponds to the optimal solution $(R^*_{x,t+1}, R^*_{y,t+1})$ with external market price ε_{t+1} which maximizes:

$$(R_{x,t+1} - R_{x,t}) + (R_{y,t+1} - R_{y,t})\varepsilon_{t+1}. \tag{6.8}$$

Let this quantity be

$$L = (R^*_{x,t+1} - R_{x,t}) + (R^*_{y,t+1} - R_{y,t})\varepsilon_{t+1}. \tag{6.9}$$

In Diamond, a player trying to move the reserves of Φ to $(R'_{x,t+1}, R'_{y,t+1})$ only receives $(1-\beta)(R'_{x,t+1} - R_{x,t})$ while giving $(1-\beta)(R'_{y,t+1} - R_{y,t})$ to Φ. Thus, an arbitrageur wants to find the values of $(R'_{x,t+1}, R'_{y,t+1})$ that maximize:

$$(1-\beta)(R'_{x,t+1} - R_{x,t}) + (1-\beta)(R'_{y,t+1} - R_{y,t})\varepsilon_{t+1} + \mathbb{E}(\text{conversion}). \quad (6.10)$$

where $\mathbb{E}(\text{conversion})$ is the per-block amortized expectancy of the conversion operation for the arbitrageurs. From Lemma 6.5.3, we know $\mathbb{E}(\text{conversion}) \geq 0$ for Φ. This implies the arbitrageur's max gain is less than:

$$(1-\beta)(R'_{x,t+1} - R_{x,t}) + (1-\beta)(R'_{y,t+1} - R_{y,t})\varepsilon_{t+1}, \quad (6.11)$$

for the $(R'_{x,t+1}, R'_{y,t+1})$ maximizing Equation 6.10. From Equation 6.9, we know this has a maximum at $(R'_{x,t+1}, R'_{y,t+1}) = (R^*_{x,t+1}, R^*_{y,t+1})$. Therefore, the LVR of Φ is at most:

$$(1-\beta)((R^*_{x,t+1} - R_{x,t}) + (R^*_{y,t+1} - R_{y,t})\varepsilon_{t+1}) = (1-\beta)L. \quad (6.12)$$

$\square$

6.6 Implementation

We now detail an implementation of Diamond. In our implementation, we consider block producers as arbitrageurs, with names interchangeable, with block producers in Diamond not charged protocol fees. The main focus of our implementation is ensuring user experience in a Diamond pool is not degraded compared to the corresponding CFMM pool. To this point, applying an β-discount on every Diamond pool trade is not viable. To avoid this, we only consider LVR on a per-block, and not a per-transaction basis. Given the transaction sequence, in/exclusion and priority auction capabilities of block producers, block producers can either capture the block LVR of a Diamond pool themselves, or effectively sell this right to other arbitrageurs.

From an implementation standpoint, who captures the LVR is not important, but it is the block producer who must repay the LVR of a block. To enforce this, for a Diamond pool, we check the pool state in the first pool transaction each block and take escrow from the block producer. This escrow is be used in part to pay the realized LVR of the block back to the pool. The first pool transaction also returns the collateral of the previous block producer, minus the realized LVR (computable from the difference between the current pool state and the pool state at the beginning of the previous block). To ensure the collateral covers realized LVR, each proceeding pool transaction verifies that the LVR implied by the pool state as a result of the transaction can be repaid by the deposited collateral. Our implementation is based on the following two assumptions:

1. A block producer always sets the final state of a pool to the state which maximizes the LVR.

2. The block producer realizes net profits of at least the LVR corresponding to the final state of the pool.

If the final price of the block is not the price maximizing LVR, the block producer has ignored an arbitrage opportunity. The block producer can always ignore non-block producer transactions to realize the LVR, therefore, any additional included transactions must result in greater or equal utility for the block producer than the LVR.

6.6.1 Core Protocol

The first transaction interacting with a Diamond pool Φ in every block attests to the maximum and minimum prices attained by Φ during the block, p_{max} and p_{min} respectively. We call this transaction the *pool unlock transaction*. Only one pool unlock transaction is executed per pool per block. Given a current pool price of p_0 corresponding to reserves $(R_{x,0}, R_{y,0})$, it must be that $p_{min} \leq p_0 \leq p_{max}$. As such, given a move to p_{min}, some amount $\lambda_x \geq 0$ must be returned to the Φ pool and vault by the block producer. Similarly, given a move to p_{max}, $\lambda_y \geq 0$ must be returned to the Φ pool and vault by the block producer. For a final pool price p_1 with $\Upsilon_x > 0$ tokens added to the pool and $\Upsilon_y > 0$ tokens removed (implying $p_1 > p_0$), $\beta\Upsilon_x$ tokens are removed from λ_x, while $\beta\Upsilon_y$ tokens are returned to the producer who deposited λ_x. A further υ_x tokens are removed from λ_x such that:

$$PPF(R_{x,0} - (1-\beta)\Upsilon_x - \upsilon_x, R_{y,0} + (1-\beta)\Upsilon_y) = p_1. \tag{6.13}$$

The λ_y and remainder of λ_x, if any, are returned to the producer who deposited λ_x and λ_y. Given $p_1 \leq p_0$, the same process is repeated with $\beta\Upsilon_y$ and υ_y paid to the pool and vault respectively. These amounts (λ_x, λ_y) must be deposited to the protocol contract in the pool unlock transaction.

Every proceeding user transaction interacting with Φ in the block first verifies that the implied pool move stays within the bounds $[p_{min}, p_{max}]$ specified at the start of the block. Non pool-unlock transactions are executed as they would be in the corresponding CFMM pool $CFMM(\Phi)$ **without** a β discount on the amount of tokens that can be removed. If a transaction implies a move outside of these bounds, it is not executed.

The next time a pool unlock transaction is submitted (in a proceeding block), given the final price of the preceding block was p_1 the actual amount of token x or y required to be added to the pool and vault (the $\beta\Upsilon$ and υ of the required token, as derived earlier in the section) is taken from the deposited escrow, with the remainder returned to the block producer who deposited those tokens.

Remark 6.6.1. *Setting the LVR rebate parameter too high can result in protocol censorship and/or liveness issues as certain block producers may not be equipped to frictionlessly arbitrage, and as such, repay the implied LVR to the protocol. To counteract this, the LVR rebate parameter should be reduced every block in which no transactions take place. As arbitrageurs are competing to extract LVR from the pool, the LVR rebate parameter will eventually become low enough for block producers to include Diamond transactions. After transactions have been executed, the LVR rebate parameter should be reset to its initial value. Rigorous testing of initial values and decay curves are required for any choice of rebate parameter.*

6.6.2 Conversion Protocols

The described implementations in this section assume the existence of a decentralized on-chain auction.[4]

Per-block Conversion vs. Futures

Given per-block conversion (Section 6.4.3), further deposits from the block producer are required to cover the token requirements of the conversion and collateralizing the futures. The conversions for a pool Φ resulting from transactions in a block take place in the next block a pool unlock transaction for Φ is called. Given a maximum expected percentage move over τ blocks of σ_T, and a conversion of λ_y tokens at price p, the block producer collateral must be in quantities π_x and π_y such that if the block producer is long the futures:

$$1.\ \pi_x + \pi_y \frac{p}{1+\sigma_T} \geq \lambda_y\left(p - \frac{p}{1+\sigma_T}\right), \text{ and } 2.\ \frac{\pi_x}{\pi_y} = \frac{p}{1+\sigma_T}. \tag{6.14}$$

[4]First-price sealed-bid auctions can be implemented using a commit-reveal protocol. An example of such a protocol involves bidders hashing bids, committing these to the blockchain along with an over-collaterlization of the bid, with bids revealed when all bids have been committed.

If the block producer is short the futures it must be that:

$$1.\ \pi_x + \pi_y p(1+\sigma_T) \geq \lambda_y p \sigma_T, \qquad \text{and } 2.\ \frac{\pi_x}{\pi_y} = p(1+\sigma_T). \tag{6.15}$$

The first requirement in both statements is for the block producer's collateral to be worth more than the maximum expected loss. The second requirement states the collateral must be in the ratio of the pool for the maximum expected loss (which also ensures it is in the ratio of the pool for any other loss less than the maximum expected loss). This second requirement ensures the collateral can be added back into the pool when the futures are settled.

At settlement, if the futures settle in-the-money for the block producer, tokens are removed from the pool in the ratio specified by the settlement price with total value equal to the loss incurred by the pool, and paid to the block producer. If the futures settle out-of-the-money, tokens are added to the pool from the block producer's collateral in the ratio specified by the settlement price with total value equal to the loss incurred by the block producer. The remaining collateral is returned to the block producer. The pool constant is adjusted to reflect the new balances.

Remark 6.6.2. *As converting the vault does not affect pool availability, the auctions for converting the vault can be run sufficiently slowly so as to eliminate the risk of block producer censorship of the auction. We choose to not remove tokens from the pool to collateralize the futures as this reduces the available liquidity within the pool, which we see as an unnecessary reduction in benefit to users (which would likely translate to lower transaction fee revenue for the pool). For high volatility token pairs,* τ *should be chosen sufficiently small so as to not to risk pool liquidation.*

If Diamond with conversion versus futures is run on a blockchain where the block producer is able to produce multiple blocks consecutively, this can have an adverse effect on incentives. Every time the vault is converted and tokens are re-added to the pool, the liquidity of the pool increases. A block producer with control over multiple blocks can move the pool price some of the way towards the maximal LVR price, convert the vault tokens (which has 0 expectancy from Lemma 6.5.1), increase the liquidity of the pool, then move the pool towards the maximal LVR price again in the proceeding block. This process results in a slight increase in value being extracted from the pool in expectancy compared to moving the pool price immediately to the price corresponding to maximal LVR. Although the effect on incentives is small, re-adding tokens from a conversion slowly/keeping the pool constant constant mitigates/removes this benefit for such block producers.

Periodic Conversion Auction

Every τ blocks, given η tokens in the vault, $\frac{\eta}{2}$ of these tokens are auctioned off, with bids placed in the other token. The winning bidder receives these $\frac{\eta}{2}$. The winning bid, and the remaining $\frac{\eta}{2}$ tokens in the vault are added to the pool.

6.7 Experimental Analysis

This section presents the results of several experiments, which can be reproduced using the following public repository [55]. The results provide further evidence of the performance potential of a Diamond pool versus various benchmarks. These experiments isolate the effect that different fees, conversion frequencies, daily price moves, LVR rebate parameters, and days in operation have on a Diamond pool. Each graph represents a series of random-walk simulations which were run, unless otherwise stated, with base parameters of:

- LVR rebate parameter: 0.95.
- Average daily price move: 5%.
- Conversion frequency: Once per day.
- Blocks per day: 10.
- Days per simulation: 365.
- Number of simulations per variable: 500.

Parameter Intuition

For a Diamond pool to be deployed, we expect the existence of at least one tradeable and liquid external market price. As such, many competing arbitrageurs should exist, keeping the LVR parameter close to 1. 5% is a typcial daily move for our chosen token pair. Given a daily move of 5%, the number of blocks per day is not important, as the per block expected moves can be adjusted given the daily expected move. Given a simulator constraint of 5,000 moves per simulation, we chose 10 blocks per day for a year, as opposed to simulating Ethereum over 5,000 blocks (less than 1 day's worth of blocks), as the benefits of Diamond are more visible over a year than a day.

Each graph plots the final value of the Diamond Periodic Conversion Auction pool (unless otherwise stated) relative to the final value of the corresponding Uniswap V2 pool. The starting reserve values are \$100m USDC and 76,336 ETH, for an ETH price of \$1,310, the approximate price and pool size of the Uniswap ETH/USDC pool at the time of writing [102].

Figure 6.2 compares four strategies over the same random walks. Periodic Conversion Auction and Conversion vs. Futures replicate the Diamond protocol given the respective conversion strategies (see Section 6.4). HODL (Hold-On-for-Dear-Life), measures the performance of holding the starting reserves until the end of the simulation. The final pool value of these three strategies are then taken as a fraction of the corresponding CFMM pool following that same random walk. Immediately we can see all three of these strategies outperform the CFMM strategy in all simulations (as a fraction of the CFMM pool value, all other strategies are greater than 1), except at the initial price of 1310, where HODL and CFMM are equal, as expected.

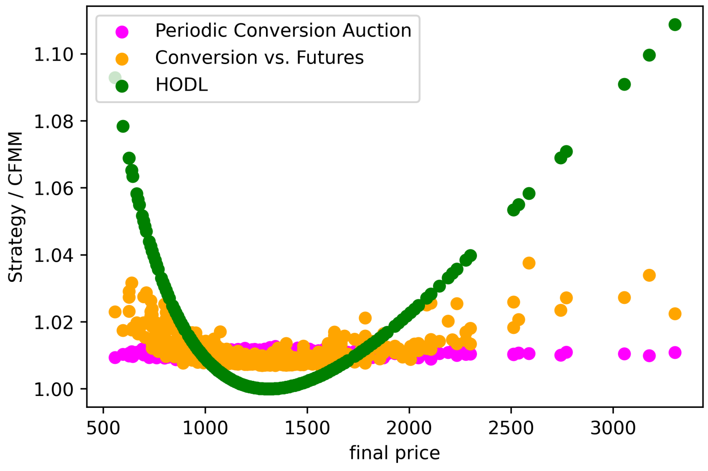

Figure 6.2: Strategy comparison.

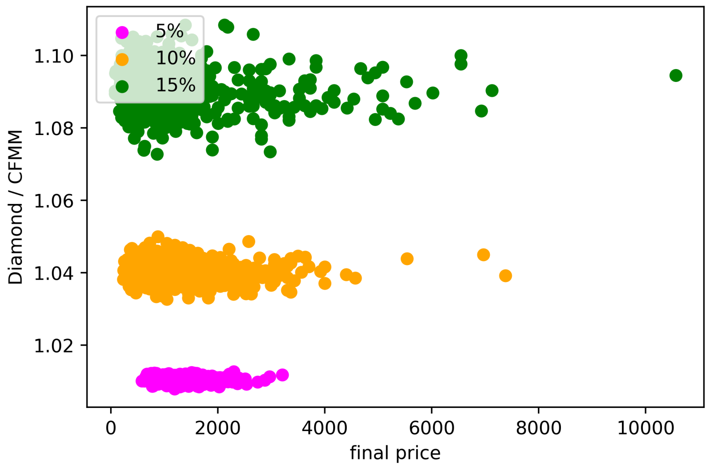

Figure 6.3: Volatility comparison.

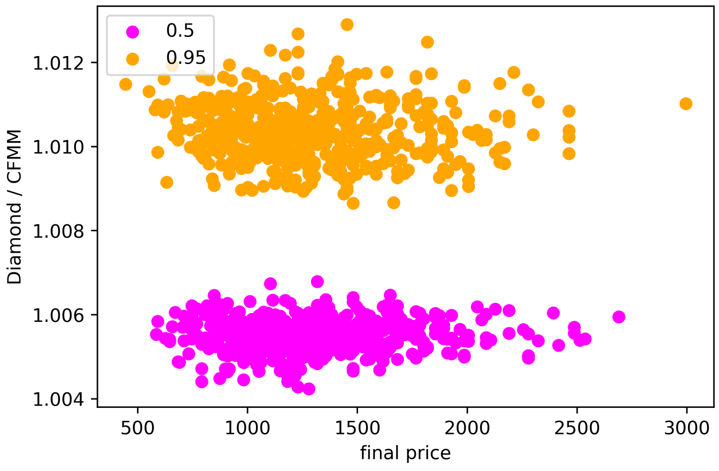

Figure 6.4: LVR rebate comparison.

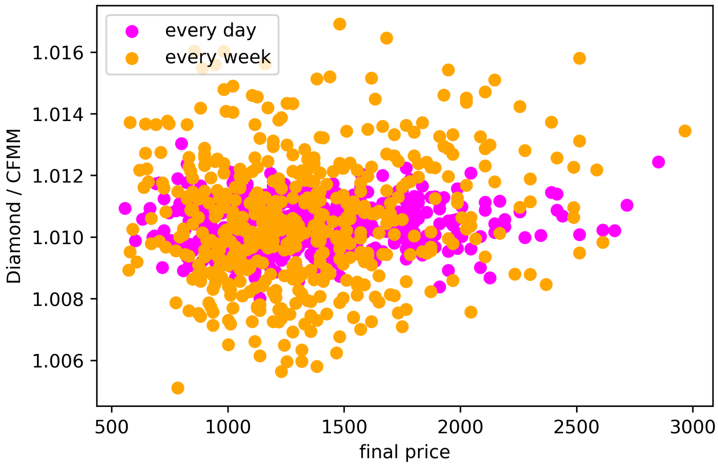

Figure 6.5: Conversion frequency comparison.

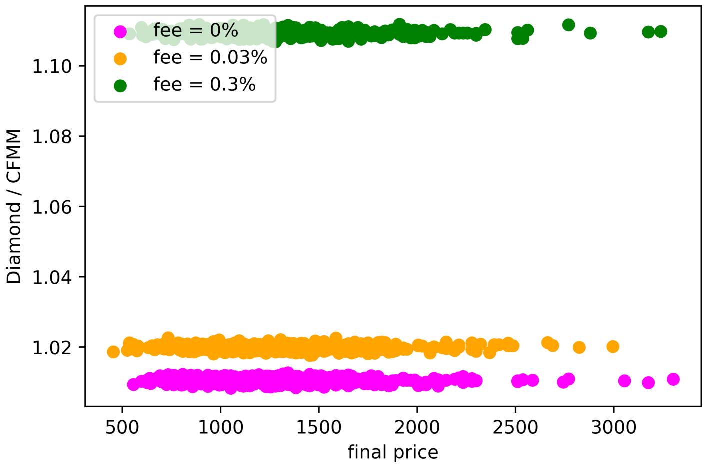

Figure 6.6: Fee comparison, given 10% of pool TVL trades per day.

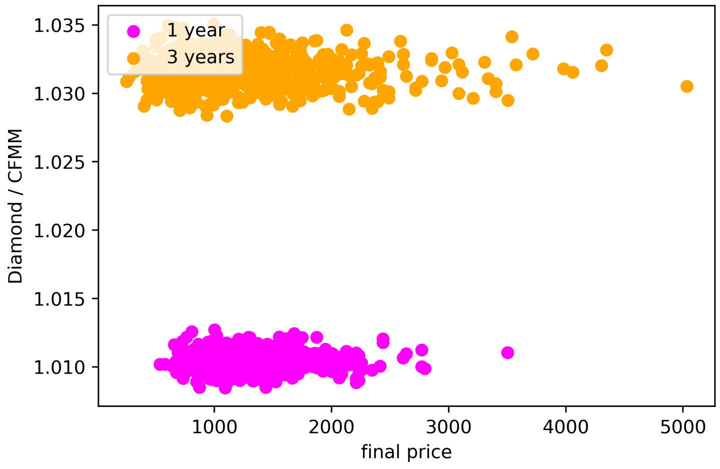

Figure 6.7: Protocol duration comparison.

The Diamond pools outperform HODL in a range around the starting price, as Diamond pools initially retain the tokens increasing in value (selling them eventually), which performs better than HODL when the price reverts. HODL performs better in tail scenarios as all other protocols consistently sell the token increasing in value on these paths. Note Periodic Conversion slightly outperforms Conversion vs. Futures when finishing close to the initial price, while slightly underperforming at the tails. This is because of the futures exposure. Although these futures have no expectancy for the protocol, they increase the variance of the Conversion vs. Futures strategy, outperforming when price changes have momentum, while underperforming when price changes revert.

Figure 6.3 identifies a positive relationship between the volatility of the price and the out-performance of the Diamond pool over its corresponding CFMM pool. This is in line with the results of [82] where it is proved LVR grows quadratically in volatility. Figure 6.4 demonstrates that, as expected, a higher LVR rebate parameter β retains more value for the Diamond pool.

Figure 6.5 shows that higher conversion frequency (1 day) has less variance for the pool value (in this experiment once per day conversion has mean 1.011234 and standard deviation 0.000776 while once per week conversion has mean 1.011210 and standard deviation 0.002233). This highlights an important trade-off for protocol deployment and LPs. Although lower variance corresponding to more frequent conversion auctions is desirable, more frequent auctions may centralize the players participating in the auctions due to technology requirements. This would weaken the competition guarantees needed to ensure that the auction settles at the true price in expectancy.

Figure 6.6 compares Diamond to the CFMM pool under the specified fee structures (data-points corresponding to a particular fee apply the fee to both the Uniswap pool and the Diamond pool) assuming 10% of the total value locked in each pool trades daily. The compounding effect of Diamond's LVR rebates with the fee income every block result in a significant out-performance of the Diamond protocol as fees increase. This observation implies that given the LVR protection provided by Diamond, protocol fees can be reduced significantly for users, providing a further catalyst for a DeFi revival. Figure 6.7 demonstrates that the longer Diamond is run, the greater the out-performance of the Diamond pool versus its corresponding CFMM pool.

6.8 Conclusion

We present Diamond, an AMM protocol which provably protects against LVR. The described implementation of Diamond stands as a generic template to address LVR in any CFMM. The experimental results of Section 6.7 provide strong evidence in support of the LVR protection of Diamond, complementing the formal results of Section 6.5. It is likely that block producers will be required to charge certain users more transaction fees to participate in Diamond pools to compensate for this LVR rebate, with informed users being charged more for block inclusion than uninformed users. As some or all of these proceeds are paid to the pool with these proceeds coming from informed users, we see this as a desirable outcome.

Given the protocol-level protections provided by Diamond, research must now focus on user protections in order to keep protocol utility in the hands of protocol contributors. Rook [95] and CoW protocol [41] have taken important steps in this regard, although many open problems still exist.

Bibliography

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient Synchronous Byzantine Consensus. https://eprint.iacr.org/2017/307, 2017. Accessed: 10/01/2023.

[2] Ittai Abraham, Danny Dolev, Ittay Eyal, and Joseph Y. Halpern. Colordag: An Incentive-Compatible Blockchain. https://eprint.iacr.org/2022/308, 2022. Accessed: 03/01/2023.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A Blockchain Protocol Based on Reconfigurable Byzantine Consensus. In James Aspnes, Alysson Bessani, Pascal Felber, and João Leitão, editors, 21st International Conference on Principles of Distributed Systems (OPODIS 2017), volume 95 of Leibniz International Proceedings in Informatics (LIPIcs), pages 25:1–25:19, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[4] Hayden Adams, River Keefer, Moody Salem, Noah Zinsmeister, and Dan Robinson. Uniswap V3 Core, 2021.

[5] Hayden Adams, Noah Zinsmeister, and Dan Robinson. Uniswap V2 Core, 2020.

[6] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Mike Dahlin, Jean-Philippe Martin, and Carl Porth. BAR Fault Tolerance for Cooperative Services. SIGOPS Oper. Syst. Rev., 39(5):45–58, October 2005.

[7] Humoud Alsabah and Agostino Capponi. Pitfalls of Bitcoin's Proof-of-Work: R&D Arms Race and Mining Centralization. https://ssrn.com/abstract=3273982, 2020. Accessed: 10/01/2023.

[8] Amazon. Amazon Web Service Revenues, Q1 2022. https://ir.aboutamazon.com/news-release/news-release-details/2022/Amazon.com-Announces-First-Quarter-Results-f0188db95/. Accessed: 18/08/2022.

[9] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Rational vs Byzantine Players in Consensus-Based Blockchains. In Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '20, page 43–51, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems.

[10] Yackolley Amoussou-Guenou, Bruno Biais, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Rational Behaviors in Committee-Based Blockchains. In Quentin Bramas, Rotem Oshman, and Paolo Romano, editors, 24th International Conference on Principles of Distributed Systems (OPODIS 2020), volume 184 of Leibniz International Proceedings in Informatics (LIPIcs), pages 12:1–12:16, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[11] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. Correctness and Fairness of Tendermint-core Blockchains. https://arxiv.org/pdf/1805.08429, 2018. Accessed: 03/01/2023.

[12] Yackolley Amoussou-Guenou, Antonella Del Pozzo, Maria Potop-Butucaru, and Sara Tucci-Piergiovanni. On Fairness in Committee-Based Blockchains. In Emmanuelle Anceaume, Christophe Bisière, Matthieu Bouvard, Quentin Bramas, and Catherine Casamatta, editors, 2nd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2020), volume 82 of Open Access Series in Informatics (OASIcs), pages 4:1–4:15, Dagstuhl, Germany, 2021. Schloss Dagstuhl–Leibniz-Zentrum für Informatik.

[13] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. An Analysis of Uniswap Markets. In Cryptoeconomic Systems Journal, 2019.

[14] Nick Arnosti and S. Matthew Weinberg. Bitcoin: A natural oligopoly. In Avrim Blum, editor, 10th Innovations in Theoretical Computer Science,

ITCS 2019, Leibniz International Proceedings in Informatics, LIPIcs, Germany, January 2019. Schloss Dagstuhl- Leibniz-Zentrum fur Informatik GmbH, Dagstuhl Publishing. Funding Information: Supported by NSF CCF-1717899.; 10th Innovations in Theoretical Computer Science, ITCS 2019 ; Conference date: 10-01-2019 Through 12-01-2019.

[15] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. Helix: A Fair Blockchain Consensus Protocol Resistant to Ordering Manipulation. IEEE Transactions on Network and Service Management, 18(2):1584–1597, 2021.

[16] Sarah Azouvi and Alexander Hicks. Sok: Tools for Game Theoretic Models of Security for Cryptocurrencies. Cryptoeconomic Systems, 0(1), 2021. `https://cryptoeconomicsystems.pubpub.org/pub/azouvi-sok-security`.

[17] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. SoK: Consensus in the Age of Blockchains. In Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT '19, pages 183–198, New York, NY, USA, 2019. Association for Computing Machinery.

[18] Xianglin Bao, Cheng Su, Yan Xiong, Wenchao Huang, and Yifei Hu. FLChain: A Blockchain for Auditable Federated Learning with Trust and Incentive. In 2019 5th International Conference on Big Data Computing and Communications (BIGCOM), pages 151–159, 2019.

[19] Niko Baric and Birgit Pfitzmann. Collision-Free Accumulators and Fail-Stop Signature Schemes without Trees. In Proceedings of the 16th Annual International Conference on Theory and Application of Cryptographic Techniques, EUROCRYPT'97, page 480–494, Berlin, Heidelberg, 1997. Springer-Verlag.

[20] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. A Theory of Automated Market Makers in DeFi. In Ferruccio Damiani and Ornela Dardha, editors, Coordination Models and Languages, pages 168–187, Cham, 2021. Springer International Publishing.

[21] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Maximizing Extractable Value from Automated Market Makers. In Financial Cryptography and Data Security, Berlin, Heidelberg, 2022. Springer Berlin Heidelberg.

[22] Carsten Baum, Bernardo David, and Tore Frederiksen. P2DEX: Privacy-Preserving Decentralized Cryptocurrency Exchange. In Kazue Sako and Nils Ole Tippenhauer, editors, Applied Cryptography and Network Security, pages 163–194. Springer International Publishing, 2021.

[23] Eli Ben-Sasson, Alessandro Chiesa, Christina Garman, Matthew Green, Ian Miers, Eran Tromer, and Madars Virza. Zerocash: Decentralized Anonymous Payments from Bitcoin. In 2014 IEEE Symposium on Security and Privacy, pages 459–474, New York, NY, USA, 2014. IEEE Computer Society.

[24] Josh Benaloh and Michael de Mare. One-Way Accumulators: A Decentralized Alternative to Digital Signatures. In Tor Helleseth, editor, Advances in Cryptology — EUROCRYPT '93, pages 274–285, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[25] Daniel Benarroch, Matteo Campanelli, Dario Fiore, Kobi Gurkan, and Dimitris vKolonelos. Zero-Knowledge Proofs for Set Membership: Efficient, Succinct, Modular. In Nikita Borisov and Claudia Diaz, editors, Financial Cryptography and Data Security, pages 393–414, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[26] Bruno Biais, Christophe Bisière, Matthieu Bouvard, and Catherine Casamatta. The blockchain folk theorem. IDEI Working Papers 873, Institut d'Économie Industrielle (IDEI), Toulouse, 2017. Accessed: 19/05/2021.

[27] Maxim Bichuch and Zachary Feinstein. Axioms for Automated Market Makers: A Mathematical Framework in FinTech and Decentralized Finance. `https://arxiv.org/abs/2210.01227`, 2022. Accessed: 18/10/2022.

[28] Georgios Birmpas, Elias Koutsoupias, Philip Lazos, and Francisco J. Marmolejo-Cossío. Fairness and Efficiency in DAG-Based Cryptocurrencies. In Financial Cryptography and Data Security, pages 79–96, Cham, 2020. Springer International Publishing.

[29] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS'17, page 118–128, Red Hook, NY, USA, 2017. Curran Associates Inc.

[30] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, Advances in Cryptology – CRYPTO 2019, pages 561–586, Cham, 2019. Springer International Publishing.

[31] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. https://arxiv.org/abs/1807.049385, 2019. Accessed: 21/05/2021.

[32] Eric Budish. The Economic Limits of Bitcoin and the Blockchain. Working Paper 24717, National Bureau of Economic Research, June 2018.

[33] Eric Budish, Peter Cramton, and John Shim. The High-Frequency Trading Arms Race: Frequent Batch Auctions as a Market Design Response *. The Quarterly Journal of Economics, 130(4):1547–1621, 07 2015.

[34] Vitalik Buterin, Daniel Reijsbergen, Stefanos Leonardos, and Georgios Piliouras. Incentives in Ethereum's Hybrid Casper Protocol. In 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pages 236–244, Seoul, South Korea, 2019. IEEE.

[35] Agostino Capponi and Ruizhe Jia. The Adoption of Blockchain-based Decentralized Exchanges. https://arxiv.org/abs/2103.08842, 2021. Accessed: 18/10/2022.

[36] Chainlink. https://chain.link/. Accessed: 11/10/2022.

[37] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Proceedings of the 26th Annual International Conference on Advances in Cryptology, CRYPTO'06, page 78–96, Berlin, Heidelberg, 2006. Springer-Verlag.

[38] Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. Theoretical Computer Science, 777:155–183, 2019.

[39] Michele Ciampi, Muhammad Ishaq, Malik Magdon-Ismail, Rafail Ostrovsky, and Vassilis Zikas. FairMM: A Fast and Frontrunning-Resistant Crypto Market-Maker. Cryptology ePrint Archive, Report 2021/609, 2021. Accessed: 02/02/2022.

[40] Theodoros Constantinides and John Cartlidge. Block Auction: A General Blockchain Protocol for Privacy-Preserving and Verifiable Periodic Double Auctions. In 2021 IEEE International Conference on Blockchain (Blockchain), pages 513–520, United States, 2021. IEEE Computer Society.

[41] CoW Protocol. `https://docs.cow.fi/`. Accessed: 11/10/2022.

[42] Phil Daian, Rafael Pass, and Elaine Shi. Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake. In Financial Cryptography and Data Security, pages 23–41. Springer International Publishing, 2019.

[43] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges. `https://arxiv.org/abs/1904.05234`, 2019. Accessed: 19/01/2022.

[44] Sanmay Das and Malik Magdon-Ismail. Adapting to a Market Shock: Optimal Sequential Market-Making. In Proceedings of the 21st International Conference on Neural Information Processing Systems, NIPS'08, page 361–368, Red Hook, NY, USA, 2008. Curran Associates Inc.

[45] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. J. ACM, 35(2):288–323, April 1988.

[46] Etherscan. `https://etherscan.io/gastracker`. Accessed: 25/07/2022.

[47] Alex Evans, Guillermo Angeris, and Tarun Chitra. Optimal Fees for Geometric Mean Market Makers. In Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner, editors, Financial Cryptography and Data Security. FC 2021 International Workshops, pages 65–79, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[48] Ittay Eyal and Emin Gün Sirer. Majority is Not Enough: Bitcoin Mining is Vulnerable. Commun. ACM, 61(7):95–102, June 2018.

[49] Giulia Fanti, Leonid Kogan, Sewoong Oh, Kathleen Ruan, Pramod Viswanath, and Gerui Wang. Compounding of Wealth in Proof-of-Stake Cryptocurrencies. In Ian Goldberg and Tyler Moore, editors, Financial

Cryptography and Data Security, pages 42–61, Cham, 2019. Springer International Publishing.

[50] Flashbots. https://explore.flashbots.net. Accessed: 11/10/2022.

[51] Hisham S. Galal and Amr M. Youssef. Publicly Verifiable and Secrecy Preserving Periodic Auctions. In Matthew Bernhard, Andrea Bracciali, Lewis Gudgeon, Thomas Haines, Ariah Klages-Mundt, Shin'ichiro Matsuo, Daniel Perez, Massimiliano Sala, and Sam Werner, editors, Financial Cryptography and Data Security. FC 2021 International Workshops, pages 348–363, Berlin, Heidelberg, 2021. Springer Berlin Heidelberg.

[52] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In Advances in Cryptology - EUROCRYPT 2015, pages 281–310, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[53] Github. https://github.com/The-CTra1n/MarvelDC, 2022.

[54] Github. https://github.com/MEVProof/Contracts, 2022.

[55] Github. https://github.com/The-CTra1n/LVR, 2022.

[56] S Goldwasser, S Micali, and C Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing, STOC '85, page 291–304, New York, NY, USA, 1985. Association for Computing Machinery.

[57] Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, Advances in Cryptology – EUROCRYPT 2016, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[58] Lewis Gudgeon, Daniel Perez, Dominik Harz, Benjamin Livshits, and Arthur Gervais. The Decentralized Financial Crisis. In 2020 Crypto Valley Conference on Blockchain Technology (CVCBT), pages 1–15, 2020.

[59] Kobi Gurkan, Koh Wei Jie, and Barry Whitehat. Community Proposal: Semaphore: Zero-Knowledge Signaling on Ethereum, 2020. Accessed: 25/01/2022.

[60] Josojo. MEV capturing AMMs. https://ethresear.ch/t/mev-capturing-amm-mcamm/13336, 2022. Accessed: 18/10/2022.

[61] Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar Weippl. Estimating (Miner) Extractable Value is Hard, Let's Go Shopping! https://ia.cr/2021/1231, 2021. Accessed: 18/05/2022.

[62] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-Fairness for Byzantine Consensus. In Daniele Micciancio and Thomas Ristenpart, editors, Advances in Cryptology - CRYPTO 2020, pages 451–480, Cham, 2020. Springer International Publishing.

[63] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A Provably Secure Proof-of-Stake Blockchain Protocol. In Jonathan Katz and Hovav Shacham, editors, Advances in Cryptology – CRYPTO 2017, pages 357–388. Springer International Publishing, 2017.

[64] Hyesung Kim, Jihong Park, Mehdi Bennis, and Seong-Lyun Kim. Blockchained On-Device Federated Learning. IEEE Communications Letters, 24(6):1279–1283, 2020.

[65] Abhiram Kothapalli, Andrew Miller, and Nikita Borisov. Smartcast: An incentive compatible consensus protocol using smart contracts. In Andrew Miller, Michael Brenner, Kurt Rohloff, Joseph Bonneau, Vanessa Teague, Andrea Bracciali, Massimiliano Sala, Federico Pintore, Markus Jakobsson, and Peter Y.A. Ryan, editors, Financial Cryptography and Data Security - FC 2017 International Workshops, Revised Selected Papers, pages 536–552, Sliema, Malta, 2017. Springer-Verlag Berlin Heidelberg.

[66] Vijay Krishna. Auction theory. Academic press, 2009.

[67] Bhaskar Krishnamachari, Qi Feng, and Eugenio Grippo. Dynamic Automated Market Makers for Decentralized Cryptocurrency Exchange. In 2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pages 1–2, 2021.

[68] Jae Kwon. Tendermint: Consensus without mining. https://tendermint.com/static/docs, 2014. Accessed: 19/05/2021.

[69] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. FairLedger: A Fair Blockchain Protocol for Financial Institutions. In Pascal Felber, Roy Friedman, Seth Gilbert, and Avery Miller, editors, 23rd International Conference on Principles of Distributed Systems (OPODIS 2019), volume 153 of Leibniz International Proceedings in Informatics

(LIPIcs), pages 4:1–4:17, Dagstuhl, Germany, 2020. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[70] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. BAR Gossip. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, OSDI '06, page 191–204, USA, 2006. USENIX Association.

[71] Yuzheng Li, Chuan Chen, Nan Liu, Huawei Huang, Zibin Zheng, and Qiang Yan. A Blockchain-Based Decentralized Federated Learning Framework with Committee Consensus. IEEE Network, 35(1):234–241, 2021.

[72] Ziyao Liu, Jiale Guo, Wenzhuo Yang, Jiani Fan, Kwok-Yan Lam, and Jun Zhao. Privacy-Preserving Aggregation in Federated Learning: A Survey. `https://arxiv.org/abs/2203.17005`, 2022. Accessed: 30/06/2022.

[73] Ziyao Liu, Nguyen Cong Luong, Wenbo Wang, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. A Survey on Blockchain: A Game Theoretical Perspective. IEEE Access, 7:47615–47643, 2019.

[74] Stefan Loesch, Nate Hindman, Mark B Richardson, and Nicholas Welch. Impermanent Loss in Uniswap v3. `https://arxiv.org/abs/2111.09192`, 2021. Accessed: 18/10/2022.

[75] Anna Lysyanskaya and Nikos Triandopoulos. Rationality and Adversarial Behavior in Multi-party Computation. In Cynthia Dwork, editor, Advances in Cryptology - CRYPTO 2006, pages 180–197, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[76] Conor McMenamin and Vanesa Daza. Marvel DC: A Blockchain-Based Decentralized and Incentive-Compatible Distributed Computing Protocol. `https://arxiv.org/abs/2207.14011`, 2022. Accessed: 04/01/2023.

[77] Conor McMenamin, Vanesa Daza, Matthias Fitzi, and Padraic O'Donoghue. FairTraDEX: A Decentralised Exchange Preventing Value Extraction. In Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security, DeFi'22, page 39–46, New York, NY, USA, 2022. Association for Computing Machinery.

[78] Conor McMenamin, Vanesa Daza, and Bruno Mazorra. Diamonds are Forever, Loss-Versus-Rebalancing is Not. `https://arxiv.org/abs/2210.10601`, 2022. Accessed: 04/01/2023.

[79] Conor McMenamin, Vanesa Daza, and Matteo Pontecorvi. Achieving State Machine Replication without Honest Players. In Proceedings of the 3rd ACM Conference on Advances in Financial Technologies, AFT '21, page 1–14, New York, NY, USA, 2021. Association for Computing Machinery.

[80] Ralph C. Merkle. A Digital Signature Based on a Conventional Encryption Function. In Carl Pomerance, editor, Advances in Cryptology — CRYPTO '87, pages 369–378, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[81] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous Distributed E-Cash from Bitcoin. In 2013 IEEE Symposium on Security and Privacy, pages 397–411, United States, 2013. IEEE Computer Society.

[82] Jason Milionis, Ciamac C. Moallemi, Tim Roughgarden, and Anthony Lee Zhang. Quantifying Loss in Automated Market Makers. In Fan Zhang and Patrick McCorry, editors, Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security. ACM, 2022.

[83] Thomas Moscibroda, Stefan Schmid, and Roger Wattenhofer. When Selfish Meets Evil: Byzantine Players in a Virus Inoculation Game. In Proceedings of the Twenty-Fifth Annual ACM Symposium on Principles of Distributed Computing, PODC '06, pages 35–44, New York, NY, USA, 2006. Association for Computing Machinery.

[84] Thomas Moscibroda, Stefan Schmid, and Roger Wattenhofer. The Price of Malice: A Game-Theoretic Framework for Malicious Behavior. Internet Mathematics, 6(2):125–156, 2009.

[85] Satoshi Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. `https://bitcoin.org/bitcoin.pdf`, 2008. Accessed: 19/05/2021.

[86] Kevin Alarcón Negy, Peter R. Rizun, and Emin Gün Sirer. Selfish Mining Re-Examined. In Joseph Bonneau and Nadia Heninger, editors, Financial Cryptography and Data Security, pages 61–78, Cham, 2020. Springer International Publishing.

[87] Lan Nguyen. Accumulators from bilinear pairings and applications. In Proceedings of the 2005 International Conference on Topics in Cryptology, CT-RSA'05, page 275–292, Berlin, Heidelberg, 2005. Springer-Verlag.

[88] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. Algorithmic Game Theory. Cambridge University Press, Cambridge, 2007.

[89] Perpetual Powers of Tau. https://zkproof.org/2021/06/30/setup-ceremonies/. Accessed: 11/10/2022.

[90] Andreas Park. The Conceptual Flaws of Constant Product Automated Market Making. ERN: Other Microeconomics: General Equilibrium & Disequilibrium Models of Financial Markets, 2021.

[91] Rafael Pass, Lior Seeman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, Advances in Cryptology – EUROCRYPT 2017, pages 643–673, Cham, 2017. Springer International Publishing.

[92] Rafael Pass and Elaine Shi. FruitChains: A Fair Blockchain. In Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC '17, pages 315–324, New York, NY, USA, 2017. Association for Computing Machinery.

[93] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying Blockchain Extractable Value: How dark is the forest? In 2022 IEEE Symposium on Security and Privacy (SP), pages 198–214, 2022.

[94] Ioanid Roşu and Fahad Saleh. Evolution of Shares in a Proof-of-Stake Cryptocurrency. Manage. Sci., 67(2):661–672, feb 2021.

[95] Rook. https://docs.rook.fi/reference/. Accessed: 11/10/2022.

[96] Tim Roughgarden. Transaction Fee Mechanism Design. SIGecom Exch., 19(1):52–55, jul 2021.

[97] Timon Rückel, Johannes Sedlmeir, and Peter Hofmann. Fairness, Integrity, and Privacy in a Scalable Blockchain-Based Federated Learning System. Comput. Netw., 202(C), jan 2022.

[98] Fahad Saleh. Blockchain Without Waste: Proof-of-Stake. In Review of Financial Studies, volume 34, pages 1156–1190, 2021, 07 2020. March.

[99] Jakub Sliwinski and Roger Wattenhofer. Blockchains Cannot Rely on Honesty. https://disco.ethz.ch/courses/fs19/sirocco/honesty.pdf, 2020. Accessed: 21/05/2021.

[100] @thiccythot. `https://dune.com/thiccythot/uniswap-markouts`. Accessed: 11/10/2022.

[101] Kentaroh Toyoda and Allan N. Zhang. Mechanism Design for An Incentive-aware Blockchain-enabled Federated Learning Platform. In 2019 IEEE International Conference on Big Data (Big Data), pages 395–403, 2019.

[102] Uniswap. `https://app.uniswap.org/`. Accessed: 11/10/2022.

[103] Jiasi Weng, Jian Weng, Jilian Zhang, Ming Li, Yue Zhang, and Weiqi Luo. DeepChain: Auditable and Privacy-Preserving Deep Learning with Blockchain-Based Incentive. IEEE Transactions on Dependable and Secure Computing, 18(5):2438–2455, 2021.

[104] Moti Yung. The Mobile Adversary Paradigm in Distributed Computation and Systems, page 171–172. Association for Computing Machinery, New York, NY, USA, 2015.

[105] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. An Expressive (Zero-Knowledge) Set Accumulator. In 2017 IEEE European Symposium on Security and Privacy (EuroS P), pages 158–173, United States, 2017. IEEE.

Appendix A

Abbreviations and Notation

This section outlines the abbreviations and notation as used throughout the thesis.

Abbreviations

FBA: Frequent Batch Auction.

WSFBA: Width-Sensitive Frequent Batch Auction.

MM: Market Maker.

LP: Liquidity Provider.

CP: Clearing Price.

SINCE: Strong Incentive Compatible in Expectation.

SNE: Strong Nash Equilibrium.

PKI: Public Key Infrastructure.

GSR: Global Stabilization Round.

SMR: State Machine Replication.

ZK: Zero Knowledge.

NIZK: Non-Interactive Zero Knowledge.

SNARK: Succinct Argument of Knowledge.

PoK: Proof of Knowledge.

SoK: Signature of Knowledge.

FL: Federated Learning.

DC: Distributed Computing.

TTP: Trusted Third Party.

MEV: Maximal Extractable Value.

EEV: Expected Extractable Value.

AMM: Automated Market Maker.

CFMM: Constant-Function Market Maker.

DEX: Decentralized Exchange Protocol.

MPC: Multi-Party Computation.

LVR: Loss-Versus-Rebalancing.

McAMM: MEV-capturing AMM.

BAR: Byzantine-Altruistic-Rational.

Notation

$negl()$: negligible function.

κ: cryptographic security parameter.

ψ: game-theoretic security parameter.

$\mathcal{R}$: randomness.

$\mathcal{S}$: serial number.

Π: protocol.

$\mathcal{A}$: adversary.

B: block.

$\mathcal{H}$: blockchain height.

$\mathcal{X}$: action.

u: utility.

r: round.

$\mathcal{P}$: player.

V: SMR update.

t_i^r: player private information.

T_i^r: set of player private information.

str: strategy.

α: adversarial share.

$\mathcal{J}$: deviator set.

Str: strategy set.

$\mathbb{S}$: stake share.

tx: transaction.

$\equiv$: strategy equivalence relation.

Str_i^{NSD}: set of non-dominated strategies for $\mathcal{P}_i$.

$>_u$: strategy dominance relation.

Δ: bound on message delivery after GSR.

$blacklistedSNs$: blacklisted serial numbers.

ω: probability of good computations being rewarded.

γ: probability of bad computations being rewarded.

$\varkappa$: computation target value.

$f_{\varkappa}$: target function.

$\mathcal{C}$: computer.

$\mathfrak{C}$: set of computers.

ε: external market price.

δ: multiplicative market-impact coefficient.

D: net notional imbalance in a FairTraDEX auction.

Q_{not}: max notional imbalance in a FairTraDEX auction.

Z₿: notional.

p: pool price.

η, Υ, υ: token quantities.

Φ: Diamond pool.

R: pool reserves.

PnL: profit-or-loss.

PPF: pool pricing function.

PIF: pool invariant function.

τ: conversion frequency.

k: pool constant.

β: LVR-rebate parameter.

Appendix B

Publications

Conference proceedings

2021 Conor McMenamin, Vanesa Daza, & Matteo Pontecorvi. Achieving state machine replication without honest players. In AFT '21: Proceedings of the 3rd ACM Conference on Advances in Financial Technologies.

Abstract. Existing standards for player characterization in tokenized state machine replication protocols depend on honest players who will always follow the protocol, regardless of possible token increases for deviating. Given the ever-increasing market capitalization of these tokenized protocols, honesty is becoming more expensive and more unrealistic. As such, this out-dated player characterization must be removed to provide true guarantees of safety and liveness in a major stride towards universal trust in state machine replication protocols and a new scale of adoption. As all current state machine replication protocols are built on these legacy standards, it is imperative that a new player model is identified and utilized to reflect the true nature of players in tokenized protocols, now and into the future. To this effect, we propose the ByRa player model for state machine replication protocols. In the ByRa model, players either attempt to maximize their tokenized rewards, or behave adversarially. This merges the fields of game theory and distributed systems, an intersection in which tokenized state machine replication protocols exist, but on which little formalization has been carried out. In the ByRa model, we identify the properties of strong incentive compatibility in expectation and fairness that all protocols must satisfy in order to achieve state machine replication. We then provide Tenderstake, a protocol which provably satisfies these properties, and by doing so, achieves state machine replication in the ByRa model.

2022 Conor McMenamin, Vanesa Daza, Matthias Fitzi & Padraic O'Donoghue. FairTraDEX: A Decentralised Exchange Preventing Value Extraction. In DeFi'22: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security.

Abstract. We present FairTraDEX, a DEX protocol based on FBAs, which provides formal game-theoretic guarantees against extractable value. FBAs, when run by a trusted third-party, ensure that the unique game-theoretic optimal strategy for all players is to trade at the true market-implied price of the underlying token swap, excluding explicit, pre-determined fees. FairTraDEX replicates the key features of an FBA that provide these game-theoretic guarantees using a combination of set-membership in zero-knowledge protocols and an escrow-enforced commit-reveal mechanism. We extend the results of FBAs to handle monopolistic and/or malicious liquidity providers. We provide real-world examples that demonstrate that the costs of executing orders in existing academic and industry-standard protocols become prohibitive as order size increases due to basic value extraction techniques, popularized as maximal extractable value. We further demonstrate that FairTraDEX protects against these execution costs, guaranteeing a fixed fee model independent of order size, the first guarantee of it's kind for a DEX protocol. We also provide detailed Solidity and pseudo-code implementations of FairTraDEX, making FairTraDEX a novel and practical contribution.

Preprints

2022 Conor McMenamin, Vanesa Daza, & Bruno Mazorra. Diamonds are Forever, Loss-Versus-Rebalancing is Not.

Abstract. The always-available liquidity of AMMs has been one of the most important catalysts in early cryptocurrency adoption. However, it has become increasingly evident that AMMs in their current form are not viable investment options for passive liquidity providers. This is because of the cost incurred by AMMs providing stale prices to arbitrageurs against external market prices, formalized as LVR. In this paper, we present Diamond, an automated market making protocol that aligns the incentives of liquidity providers and block producers in the protocol-level retention of LVR. In Diamond, block producers effectively auction the right to capture any arbitrage that exists between the external market price of a Diamond pool, and the price of the pool itself. The proceeds of these auctions are shared by the Diamond pool and block producer in a way that is proven to remain incentive compatible for the block producer. Given the participation of competing arbitrageurs, LVR is effectively prevented in Diamond. We formally prove this result, and detail an implementation of Diamond. We also provide comparative simulations of Diamond to relevant benchmarks, further evidencing the LVR-protection capabilities of Diamond. With this new protection, passive liquidity provision on blockchains becomes rationally viable, beckoning a new age for decentralized finance.

2022 Conor McMenamin & Vanesa Daza. Marvel DC: A Blockchain-Based Decentralized and Incentive-Compatible Distributed Computing Protocol.

Abstract. Decentralized computation outsourcing should allow anyone to access the large amounts of computational power that exists in the Internet of Things. Unfortunately, when trusted third parties are removed to achieve this decentralization, ensuring an outsourced computation is performed correctly remains a significant challenge. In this paper, we provide a solution to this problem. We outline Marvel DC, a fully decentralized blockchain-based distributed-computing protocol which formally guarantees that computers are strictly incentivized to correctly perform requested computations. Furthermore, Marvel DC utilizes a reputation management protocol to ensure that, for any minority of computers not performing calculations correctly, these computers are identified and selected for computations with diminishing probability. We then outline Privacy Marvel DC, a privacy-enhanced version of Marvel DC which decouples results from the computers which computed them, making the protocol suitable for computations such as Federated Learning, where results can reveal sensitive information about that computer that computed them. We provide an implementation of Marvel DC and analyses of both protocols, demonstrating that they are not only the first protocols to provide the aforementioned formal guarantees, but are also practical, competitive with prior attempts in the field, and ready to deploy.