

Lecture Notes in Operations Research

Panos Pardalos · Ilias Kotsireas ·
Yike Guo · William Knottenbelt *Editors*

Mathematical Research for Blockchain Economy

3rd International Conference MARBLE
2022, Vilamoura, Portugal

 Springer

Lecture Notes in Operations Research

Editorial Board

Ana Paula Barbosa-Povoa, University of Lisbon, LISBOA, Portugal

Adiel Teixeira de Almeida , Federal University of Pernambuco, Recife, Brazil

Noah Gans, The Wharton School, University of Pennsylvania, Philadelphia, USA

Jatinder N. D. Gupta, University of Alabama in Huntsville, Huntsville, USA

Gregory R. Heim, Mays Business School, Texas A&M University, College Station, USA

Guowei Hua, Beijing Jiaotong University, Beijing, China

Alf Kimms, University of Duisburg-Essen, Duisburg, Germany

Xiang Li, Beijing University of Chemical Technology, Beijing, China

Hatem Masri, University of Bahrain, Sakhir, Bahrain

Stefan Nickel, Karlsruhe Institute of Technology, Karlsruhe, Germany

Robin Qiu, Pennsylvania State University, Malvern, USA

Ravi Shankar, Indian Institute of Technology, New Delhi, India

Roman Slowiński, Poznań University of Technology, Poznan, Poland

Christopher S. Tang, Anderson School, University of California Los Angeles, Los Angeles, USA

Yuzhe Wu, Zhejiang University, Hangzhou, China

Joe Zhu, Foisie Business School, Worcester Polytechnic Institute, Worcester, USA

Constantin Zopounidis, Technical University of Crete, Chania, Greece

Lecture Notes in Operations Research is an interdisciplinary book series which provides a platform for the cutting-edge research and developments in both operations research and operations management field. The purview of this series is global, encompassing all nations and areas of the world.

It comprises for instance, mathematical optimization, mathematical modeling, statistical analysis, queueing theory and other stochastic-process models, Markov decision processes, econometric methods, data envelopment analysis, decision analysis, supply chain management, transportation logistics, process design, operations strategy, facilities planning, production planning and inventory control.

LNOR publishes edited conference proceedings, contributed volumes that present firsthand information on the latest research results and pioneering innovations as well as new perspectives on classical fields. The target audience of LNOR consists of students, researchers as well as industry professionals.


Panos Pardalos · Ilias Kotsireas · Yike Guo ·
William Knottenbelt
Editors

Mathematical Research for Blockchain Economy


3rd International Conference MARBLE 2022,
Vilamoura, Portugal

 Springer

Editors

Panos Pardalos 
Department of Industrial
and Systems Engineering
University of Florida
Gainesville, FL, USA

Yike Guo 
Data Science Institute
Imperial College London
London, UK

Ilias Kotsireas 
CARGO Lab
Wilfrid Laurier University
Waterloo, ON, Canada

William Knottenbelt 
Department of Computing
Imperial College London
London, UK

ISSN 2731-040X

ISSN 2731-0418 (electronic)

Lecture Notes in Operations Research

ISBN 978-3-031-18678-3

ISBN 978-3-031-18679-0 (eBook)

<https://doi.org/10.1007/978-3-031-18679-0>

© The Editor(s) (if applicable) and The Author(s), under exclusive license to Springer Nature Switzerland AG 2023

This work is subject to copyright. All rights are solely and exclusively licensed by the Publisher, whether the whole or part of the material is concerned, specifically the rights of translation, reprinting, reuse of illustrations, recitation, broadcasting, reproduction on microfilms or in any other physical way, and transmission or information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed.

The use of general descriptive names, registered names, trademarks, service marks, etc. in this publication does not imply, even in the absence of a specific statement, that such names are exempt from the relevant protective laws and regulations and therefore free for general use.

The publisher, the authors, and the editors are safe to assume that the advice and information in this book are believed to be true and accurate at the date of publication. Neither the publisher nor the authors or the editors give a warranty, expressed or implied, with respect to the material contained herein or for any errors or omissions that may have been made. The publisher remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

This Springer imprint is published by the registered company Springer Nature Switzerland AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

Preface

This volume presents the proceedings of the 3rd International Conference on Mathematical Research for Blockchain Economy (MARBLE 2022) that was held in Vilamoura, Portugal, from July 12 to 14, 2022.

Thankfully, the gradual global recovery from the COVID-19 pandemic allowed us to return to a physical format after 2 years and to put together an exciting programme of research papers, keynote talks and a tutorial, in line with MARBLE's goal to provide a high-profile, cutting-edge platform for mathematicians, computer scientists and economists to present the latest advances and innovations related to the quantitative and economic aspects of blockchain technology. In this context, the Technical Programme Committee has accepted 14 research papers for publication and presentation on themes including mining incentives, game theory, decentralised finance, central government digital coins and stablecoins, automated market makers, blockchain infrastructure and security. The technical programme also features keynotes by the following distinguished speakers: Michael Zargham (BlockScience and Metagovernance Project), Ioana Surpateanu (Multichain Asset Managers Association), Cathy Mulligan (Instituto Superior Técnico), Rob Hygate (eWATER Services), Arda Akartuna (Elliptic and University College London), Raphaël Mazet (SimpleFi) and Stefania Barbaglio (Cassiopeia Services) as well as a tutorial on Decentralised Finance presented by Lewis Gudgeon, Daniel Perez, Sam Werner and Dragos Ilie (Imperial College London).

We thank all authors who submitted their innovative work to MARBLE 2020. In addition, we thank all members of the Technical Programme Committee and other reviewers, everyone who submitted a paper for consideration; the General Chairs, Prof. William Knottenbelt, Prof. Yike Guo and Prof. Panos Pardalos; the Organisation Chair, Jas Gill; the Web Chair, Kai Sun; the Publication Chair, Ilias Kotsireas; the Finance Chair, Diana O'Malley; the Publicity Chair, Sam Werner; and other members of the Centre for Cryptocurrency Research and Engineering who have

contributed in many different ways to the organisation effort, particularly Katerina Koutsouri. Finally, we are grateful to our primary sponsor, the Brevan Howard Centre for Financial Analysis, for their generous and ongoing support.

Vilamoura, Portugal
July 2022

Panos Pardalos
Ilias Kotsireas
Yike Guo
William Knottenbelt

Contents

Towards Equity in Proof-of-Work Mining Rewards	1
Rami A. Khalil and Naranker Dulay	
Market Equilibria and Risk Diversification in Blockchain Mining Economies	23
Yun Kuen Cheung, Stefanos Leonardos, Shyam Sridhar, and Georgios Piliouras	
On the Impact of Vote Delegation	47
Hans Gersbach, Akaki Mamageishvili, and Manvir Schneider	
Decentralized Governance of Stablecoins with Closed Form Valuation	59
Lucy Huo, Aariah Klages-Mundt, Andreea Minca, Frederik Christian Münter, and Mads Rude Wind	
Griefing Factors and Evolutionary In-Stabilities in Blockchain Mining Games	75
Stefanos Leonardos, Shyam Sridhar, Yun Kuen Cheung, and Georgios Piliouras	
Data-Driven Analysis of Central Bank Digital Currency (CBDC) Projects Drivers	95
Toshiko Matsui and Daniel Perez	
Dissimilar Redundancy in DeFi	109
Daniel Perez and Lewis Gudgeon	
DeFi Survival Analysis: Insights into Risks and User Behaviors	127
Aaron Green, Christopher Cammilleri, John S. Erickson, Oshani Seneviratne, and Kristin P. Bennett	

Gas Gauge: A Security Analysis Tool for Smart Contract Out-of-Gas Vulnerabilities 143
Behkish Nassirzadeh, Huaiying Sun, Sebastian Banescu, and Vijay Ganesh

Tweakable S_{leeve} : A Novel S_{leeve} Construction Based on Tweakable Hash Functions 169
David Chaum, Mario Larangeira, and Mario Yaksetig

Interhead Hydra: Two Heads are Better than One 187
Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka

Prediction Markets, Automated Market Makers, and Decentralized Finance (DeFi) 213
Yongge Wang

Wombat—An Efficient Stableswap Algorithm 233
Jen Hounq Lie, Tony W. H. Wong, and Alex Yin-ting Lee

Multi-Tier Reputation for Data Cooperatives 253
Abiola Salau, Ram Dantu, Kirill Morozov, Kritagya Upadhyay, and Syed Badruddoja

Towards Equity in Proof-of-Work Mining Rewards



Rami A. Khalil and Naranker Dulay

Abstract This work targets unfairness in Nakamoto, Bitcoin’s distinguished permissionless consensus protocol, towards miners with relatively small computational powers, and miners which participate during relatively unrewarding mining periods. We propose a set of computationally-grounded metrics for measuring miner expenditures, miner compensation, and coin value. Using our metrics, we quantitatively bring to light the sources of inequity in Nakamoto using Bitcoin as a real-world example. Furthermore, we propose a set of reward issuance constraints for mining incentive mechanisms to achieve equitable rewards, and argue for the efficacy of applying our constraints.

Keywords Proof-of-Work · Cryptocurrency · Rewards

1 Introduction

The most prominent Proof-of-Work (PoW) mining cryptocurrency, Bitcoin [6], does not treat all of its miners fairly. Specifically, miners with relatively small computational powers, and miners which participate in the protocol during relatively unrewarding late periods, are forced to expend more time and resources than others to create the same amount of coins. This lack of fairness is inherent to Bitcoin’s protocol design.

One of the primary tenants of Bitcoin is to remain permissionless, allowing anyone to attain a copy of the entire ledger, and insert transactions into it while remaining in agreement with all other miners on its exact contents. This permissionless agreement between all Bitcoin ledger replicas is enabled by the Nakamoto consensus

R. A. Khalil (✉) · N. Dulay
Imperial College London, London, United Kingdom
e-mail: rami.khalil@imperial.ac.uk

N. Dulay
e-mail: n.dulay@imperial.ac.uk

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_1

protocol, the first such protocol to enable consensus between a peer-to-peer network of mutually distrusting components.

PoW mining in Nakamoto is used to drive a periodic leader-election process, where the probability of a miner winning in each election period is equal to that miner's relative share of the combined mining power of all participating miners. The elected leader in each round gets to decide the next batch of coin transfers, commonly referred to as a Block of transactions, to be executed in the ledger by all replicas. Unfortunately, Nakamoto equates successful leader election with reward acquisition, awarding only leaders with new coins and transaction fees, and leaving all other unelected miners without compensation. Consequently, Nakamoto grants a miner access to rewards only in the form of an opportunity that is equal in success rate to the miner's relative share of the entire network's mining power, which all but completely eliminates all hope and incentive for very small miners to earn any coins.

Furthermore, Nakamoto provisions only a fixed number of newly issued coins per block. This number of newly issued coins is additionally periodically reduced. In Bitcoin, 50 coins were initially set as a block reward, and this reward was set to be split in half every 210,000 blocks, and sitting currently at 6.25 coins per block. Unjustifiably, this rate of periodically equal reward issuance does not respond in any form to the amount of computational power miners invest into the network, and is only implemented to promote an artificial form of coin supply scarcity. Most alarmingly, this has allowed early miners to garner a very significant number of coins in exchange for an insignificant amount of mining compared to today's mining requirements.

In retrospect, Nakamoto provides no means for miners with relatively small computational capabilities to attain the same level of reward stability as larger miners. Moreover, even without periodic exponential reward reduction, miners which participate during periods of relatively high PoW difficulty in the protocol's lifetime are forced to settle for rewards of relatively smaller value than those received by miners who participated during relatively lower PoW difficulty periods.

Scope. We only discuss specific notions of inequity in mining. Namely, inequities due to real-world conditions, such as laws, regulations, or resource costs, are not considered. Instead, the types of inequity focused on in this paper are those due to protocol specific rules that force miners which participate in the protocol to pay a higher cost per coin than others due to, for example, participating with a relatively smaller amount of computational power, or at a relatively earlier or later point in time, than other miners. Notably, how miners are rewarded, what hashing function is used, or how block creation conditions are set, is out of the scope of this paper.

Contributions. In this work, we identify, explain, and describe a solution for the sources of inequity in PoW mining against relatively small, and relatively late, miners. Primarily, we argue that the scarcity-enforcement method by which state-of-the-art PoW mining reward schemes prevent any sudden substantial growth in their coin supplies is the unnecessary reason miners are disadvantaged. More precisely, we make the following contributions:

- We introduce a computationally-grounded framework for measuring miner reward value.
- We model mining rewards in Bitcoin using our computationally-grounded framework, quantitatively demonstrating the points of inequity.
- We propose a set of reward constraints based on our computationally-grounded framework that regulate coin supply growth while preventing miner reward inequity, and argue for their effectiveness.

Structure. The remainder of this paper proceeds as follows. In Sect. 2, we review related work and literature on PoW mining rewards. Subsequently, in Sect. 3 we introduce a computationally-grounded framework for establishing a valuation metric for coins. Then, in Sect. 4 we present current sources of inequity in state-of-the-art PoW mining reward schemes, and quantify their effects on mining in Bitcoin. We then introduce in Sect. 5 a set of constraints for achieving equitable rewards, and argue for their efficacy. Lastly, we conclude this paper in Sect. 6 with a summary of our work and future work.

2 Background & Related Work

The design and analysis of protocols for permissionless distributed ledgers has garnered a torrent of attention since the recognition of Bitcoin, leading researchers and practitioners to produce a plethora of scientific work and literature. In this section we review works which, to the best of our knowledge, most relate to the understanding, design, and analysis of Nakamoto and our reward constraints. In Sect. 2.1, we focus on how mining protocols incentivize miners, assuming they are rational agents that aim to maximize their own gains, to remain compliant with protocol specifications. Subsequently, we review several analyses of mining rewards in Sect. 2.2.

2.1 Reward Protocols

Primarily, the main reward offered by all reviewed protocols comes in the form of coins within the distributed ledger in favor of miners. These rewards are intended as compensation for the miners' use of computational resources to extend the canonical chain and maintain stable consensus on the ledger contents. Inversely, miners which create divergent branches are not meant to receive rewards because they impede the stability of the consensus process.

However, identifying with certainty which divergences were intentional or not is not always practically possible, and therefore protocols cannot decisively consider every branching as non-compliant. Consequently, many state-of-the-art protocols take coarse approaches to issuing rewards and punishments. In this section, we

review only the state-of-the-art in granting coins to compliant components, leaving punishments out of scope.

The fundamental reward scheme introduced in Nakamoto is to issue a *block reward* to the miner which successfully finds a block. This reward creates new coins and credits them to the miner within the block that it created. Additionally, a secondary reward, in the form of *transaction fees*, is granted to miners that create blocks. These fees are specified and paid by the issuers of the transactions which are included in each block. All fees paid by a block's transactions are credited to the miners that created the block, after being deducted from the balances of the transaction issuers. Notably, both rewards are only valid within the chain that contains the block that issues the rewards.

Lenient Protocols. However, as stale blocks are inevitable, such a stringent policy does not compensate compliant miners that inadvertently create stale blocks. A more lenient strategy is to issue *uncle rewards* to miners which have created blocks that recently went stale [9]. These rewards are issued in the canonical chain when it is extended by a block that proves the existence of one or more such recent stale blocks. Uncle rewards are split amongst the miner which extends the canonical chain with a block that references new stale blocks as uncles, and the miners which originally produced those stale blocks.

Inclusive Protocols. Another, more general, strategy is to reward more than one miner for the creation of a block, leading to a more inclusive scheme with a smaller gap in reward stability between smaller and larger miners. The rationale behind this is that the reward distribution per block should be more representative of the block creation resource expenditure exhibited by miners throughout the search for a new block, and the rewards should not just be solely distributed to the miner which found the block's nonce in a "winner-takes-all" fashion. Several notable works have pursued this direction.

- In FruitChains [8], Pass and Shi quantify reward distribution fairness and use weak mining targets, named Fruits, to reward miners with coins if the fruits are included in a block on time. Their work reduces the discrepancy between the variability of rewards of miners with large resources and of those with smaller resources.
- Szalachowski et al. propose StrongChain [10], which uses weak mining targets to increase the weight of a chain and distribute block rewards amongst more miners by permitting the inclusion of headers of "weak" blocks in "strong" blocks. This approach also reduces the per-block reward variance of miners with smaller resources.
- Bissias and Levine generalize the mining target to be the average of the targets met by several miners in Bobtail [1], lowering the variance of inter-block arrival times and per-block rewards for miners utilizing resource amounts of all sizes.

While effective in many ways, the aforementioned works still distribute rewards in a lottery-like fashion, where only a limited number of miners may win a fixed-size pool of rewards per block. Such approaches only enable a competitive process for

miners to earn coins, and does not adjust the amounts of the coin rewards based on the amount of block creation resources invested into the process by the miners.

Decoupling Protocols. Other designs take an alternative approach, where creating new coins is a separate operation performed as a transaction in the blockchain.

- Dong and Boutaba propose publishing a challenge on a decentralized ledger and subsequently computing a Proof-of-Sequential-Work [7] (PoSeq) on that challenge to mint a new coin within a limited time-span in Elasticoin [3]. This design does permit coin rewards to increase in proportion to the amount of computational resources used by miners to create PoSeqs, but only to a limited extent. This is because the Elasticoin design assumes that the decentralized ledger has sufficient bandwidth to confirm all PoSeqs within the allotted time before they expire.
- In Melmint [4], Dong and Boutaba utilize Elasticoin minting to peg a cryptocurrency to the value of “one day of sequential computation on an up-to-date processor” and adjust its circulating supply depending on demand for it. To stabilize the currency’s value, a limited-time auction is operated on a decentralized ledger, which still may not have sufficient bandwidth to enable full network participation in a timely manner.

Responsive Protocols. On the other hand the Ergon protocol [11] issues per-block coin rewards that responsively increase in proportion to the block’s mining difficulty. However, Ergon attempts to peg the cost of production of its currency to the amount of energy miners consume, and consequently implements a periodic reward reduction mechanism similar to Bitcoin’s halving schedule [12] that is designed to make the same amount of rewards more difficult to attain over time. While this reward scheme introduces a form of proportionality between mining power and rewards, it still suffers from the drawbacks of winner-takes-all dynamics, and artificial scarcity.

To date, all state-of-the-art reward mechanisms do not scale up the potentially redeemable rewards in direct proportion to the total amount of resources utilized by miners, and instead encourage miners to compete for an increasingly scarce of constant amount of reward coins per block.

2.2 Reward Analyses

In this section we review analyses which focus on the abstract reward distribution policies employed in blockchains. With no grounded implementation in mind, the following works explore the theoretical limits of blockchain reward schemes, providing insights into the individual miner behaviors that different rewards incentivize, and the resulting network structures such incentives sustain.

Chen et al. [2] analyse block reward allocation schemes and propose a set of axioms which evaluate how the schemes perform. First, they consider a block reward scheme as symmetric if it does not vary rewards based on the identity or ordering of a miner. Second, they consider a scheme to achieve budget-balance, if the sum of all

expected rewards per miner per block does not exceed 1. Third, a scheme achieves Sybil-proofness if splitting hashing power across different identities does not yield more rewards than dedicating that power to one identity. Lastly, a scheme achieves collusion-proofness if forming coalitions of miners does not yield more rewards for the coalition than the sum of rewards of the independent miners. Weaker and stronger variations of the aforementioned axioms are also presented in [2], but they are omitted from this paper for brevity. They demonstrate that Bitcoin’s proportional allocation scheme satisfies these criteria. However, this analysis is based on long-term behavior of Bitcoin rewards, i.e. everything is well defined and balanced only on-the-long-run.

Kwon et al. [5] quantify the decentralization of PoW, PoS and DPoS consensus protocols. In their work, they quantify the decentralization of a network using the difference in combined resources between the most powerful miners, and the remaining miners which represent a certain percentile of a known number of miners. With this measure in mind, they introduce a set of constraints for reward schemes to be able to, with high probability, reach a state of full decentralization, such that the aforementioned difference in mining resources is negligible. They argue that systems without Sybil costs fail to promote decentralization, regardless of whether their consensus protocol is based on computational work or stake. Their results imply that current reward schemes are not inherently designed to encourage decentralization, since the cost of mining using fragmented mining power across multiple identities is not less than that of mining using a single identity. However, their analysis is based on a competitive coin-creation process, where miners compete for the next set of newly issued coins, and the miners which reinvest more of their earnings into attaining more mining power are destined to control a majority of network resources. Whether such a dilemma exists in a less competitive coin creation process, such as those of the decoupling and responsive protocols from Sect. 2.1, is unclear.

Thus far, mining reward scheme analyses have provided valuable insights into many aspects of existing reward mechanisms. However, a gap in knowledge exists on how *coin value* is truly distributed between miners under different reward schemes.

3 Computational Coinage Framework

In this section we present our computationally-grounded framework for establishing a valuation of a PoW cryptocurrency’s coin supply based solely on the amount of hashing power dedicated to mining it. The first goal of this framework is to establish the criteria under which we measure coin production costs. The second goal is to introduce metrics for quantifying the value of the new coins awarded in exchange for mining expenditures, without consideration for auxiliary sources of compensation, such as transaction fees. To achieve our two goals, we introduce metrics of our computationally-grounded framework according to the following criteria.

Expenditures. The primary expenditure we consider in this computationally-grounded framework is the hash-function calculation used to find valid PoWs,

which abstracts away the finer details of real-world resource requirements of this search, and accounts only for the number of hash calculations that the usage of resources results in, and the time required to finish the calculations. This purely hash-based approach was taken to establish a computationally-grounded framework for analysis, in the sense that no external variables which affect the real-world resources required to compute a hash are considered. To elaborate, we do not account for market variables such as electricity prices, hardware cost, maintenance fees, or any similar expenses.

Compensation. We consider only newly minted coins as the primary form of compensation in exchange for hashing expenditures. Similar to the abstraction of real-world costs using hash-function calculations, coins also abstract away any valuation metrics external to a blockchain, such as the coin’s exchange rate or purchasing power. Expressing the amount of compensation which a reward scheme issues is our second requirement for establishing a computationally-grounded coin value.

Value. By combining the expenditure and compensation metrics, we establish a purely hash-based valuation unit for coins suitable for our computationally-grounded framework. This approach aims to base the measurement of the value of coins purely on the number of hashing computations dedicated to creating them, enabling the quantification of the hash-based value that can be earned through mining some number of coins.

3.1 Miner Metrics

In this section, beginning from a single miner’s local perspective, we first define what a local miner represents in our computationally-grounded framework, and then present the relevant metrics for a miner.

Definition 1 (*Miner m_μ*) A miner, denoted by m_μ , where μ is some unique identifier, is characterized by (i) its absolute mining power, denoted by $power(m_\mu)$, in hashes per second, and (ii) its reward issuance difficulty, denoted by $difficulty(m_\mu)$, in expected number of hashes.

We consider a single miner in our computationally-grounded framework as a component with a certain hashing throughput, and an expected number of hashes to perform before being rewarded, as per Definition 1.

For example, let m_{4050} be a miner with $power(m_{4050}) = 2^{40}$ hashes per second that meets the reward issuance difficulty once every $difficulty(m_{4050}) = 2^{50}$ hash computations on average. This simple definition of a miner m_μ is the basis on which we build the remainder of the metrics in this section.

Given a miner m_μ , the expected length of the period of time between each reward issuance to the miner, denoted by $period(m_\mu)$, can be derived using Eq. 1.

$$period(m_\mu) = \frac{difficulty(m_\mu)}{power(m_\mu)} \quad (1)$$

For example, m_{4050} is expected to be issued a reward every $2^{50} \div 2^{40} = 1024$ s on average.

Definition 2 (*Average Hash-Time-to-Issuance $hti(m_\mu)$*) The average hash-time-to-issuance metric, denoted by $hti(m_\mu)$, represents the average amount of time before a miner's hash computation is rewarded.

In addition to the reward period, we establish in our framework a measurement of the time between completing each hash calculation, and receiving a reward as compensation for it. For every miner m_μ the average hash-time-to-issuance $hti(m_\mu)$ quantifies the average amount of waiting time for m_μ that is associated with every hash computation before the reward issuance difficulty is met, as per Definition 2. This is slightly different from the average amount of time m_μ has to wait before meeting the issuance difficulty, because the average waiting time in $hti(m_\mu)$ is accounted for *per hash* rather than *per difficulty* hashes.

Assuming that the miner's hashing throughput is uniformly sustained, which will always be assumed to be the case in the remainder of this paper, $hti(m_\mu)$ is calculated using Eq. 2.

$$hti(m_\mu) = \frac{difficulty(m_\mu) - 1}{2 \times power(m_\mu)} \quad (2)$$

As an example, consider the miner m_{13} , where $power(m_{13}) = 1$ hash per second, and $difficulty(m_{13}) = 3$ hashes on average. Using Eq. 2, $hti(m_{13}) = 1$. This can be derived by examining the waiting time expected to be incurred after performing each hash computation as follows:

1. After the first hash computation, m_{13} has to spend two more seconds computing two more hashes on average before meeting the issuance difficulty, and so the waiting time incurred after computing the first hash is 2 s.
2. After the second computation, m_{13} spends 1 more second computing one more hash on average.
3. Lastly, m_{13} is expected to have met its issuance difficulty right after the third computation without any additional waiting time.

The total waiting times divided by the number of hashes is equal to $\frac{2+1+0}{3} = 1 = hti(m_{13})$, meaning that the average time the miner waited between computing each hash and receiving a reward is 1 s, which is different from the expected waiting time of 3 s for each reward issuance. However, for much larger numbers, the metric can be approximated as $hti(m_\mu) \approx \frac{period(m_\mu)}{2}$ with negligible error.

As previously stated, the reasoning behind introducing $hti(m_\mu)$ is to quantify the average delay between miners' hash expenditures, and the reception of rewards. This metric will prove useful later on in this paper when describing the opportunity cost aspect associated with the costs of creating coins.

Definition 3 (*Reward* $reward(m_\mu)$) The reward, denoted by $reward(m_\mu)$, is the expected number of coins received by a miner m_μ that finds a PoW that meets its reward issuance difficulty.

For a single miner m_μ , we denote in our framework the reward received by m_μ as $reward(m_\mu)$ to express a miner’s compensation in coins, as per Definition 3. For example, for all Bitcoin miners, $reward(m_\mu) = 625,000,000$ coins¹ (satoshis).

Definition 4 (*Difficulty-to-Reward Ratio* $drr(m_\mu)$) The Difficulty-to-Reward Ratio, denoted by $drr(m_\mu)$, is the average number of hashes computed per reward coin for a miner.

Furthermore, for a miner m_μ , we establish the Difficulty-to-Reward Ratio as a valuation metric for the average cost of a coin in hashes, as per Definition 4. Equation 3 defines the formula for $drr(m_\mu)$.

$$drr(m_\mu) = \frac{difficulty(m_\mu)}{reward(m_\mu)} \quad (3)$$

As an example, consider miner m_{4050} from before, with $reward(m_{4050}) = 2^{30}$ coins. Using Eq. 3, the hash-based cost per coin for m_{4050} is equal to $drr(m_{4050}) = 2^{50} \div 2^{30} = 1,048,576$ hashes per coin on average.

3.2 Blockchain Metrics

In this section we introduce metrics for a chain of PoW-based blocks, rather than a single miner as in the previous section. Similarly, we first present the definition of a blockchain, followed by its associated metrics.

Definition 5 (*Blockchain* B_β) A blockchain, denoted by B_β , where β is some unique identifier, is the product of a network of miners participating in a PoW protocol.

Our computationally-grounded framework’s concept of a blockchain is devoid of implementation details, and is only constructed to enable the expression of a select few metrics of interest, as per Definition 5.

Definition 6 (*Chain Hashcap* $hashcap(B_\beta)$) The hashcap of a chain of blocks, denoted by $hashcap(B_\beta)$, is an estimate of the expected total number of hash function calculations performed by miners to create the chain.

From a blockchain viewpoint, the aggregated number of hash function calculations performed by miners to create a blockchain is referred to as the chain’s *hashcap*,² as

¹ This reward is scheduled to be halved to 312,500,000 after the current batch of 210,000 blocks is mined.

² The naming of the term *hashcap* is derived from the word Market Capitalization, or Marketcap for short, which is used in cryptocurrency markets to represent the total theoretical value of an entire supply of coins based on the coin’s market price in another currency.

per Definition 6. As the hashcap value provides an indication of the miners' expenditures towards creating a blockchain, we use it in this paper as the computationally-grounded valuation metric of the hash-based cost of production of a blockchain.

The hashcap metric value can be estimated for existing state-of-the-art PoW blockchains, such as Bitcoin, by summing the difficulty parameter for each block in a chain. However, while some hash functions can be implemented very efficiently in hardware as ASICs, some are designed to operate efficiently only on general purpose hardware such as commercially available CPUs. Consequently, the hashcaps of two blockchains which use two different hashing functions are not directly comparable.

Definition 7 (*Coinage* $coinage(B_\beta)$) The coinage of a chain of blocks, denoted by $coinage(B_\beta)$, is the total number of coins rewarded to miners in the chain.

We refer to the total supply of coins issued as rewards in a blockchain B_β in this framework as the chain's coinage, as per Definition 7. This term is also known as the circulating coin supply, or coin supply for short, in cryptocurrency markets.

Definition 8 (*Hashcap-to-Coinage Ratio* $hcr(B_\beta)$) The Hashcap-to-Coinage Ratio for a blockchain, denoted by $hcr(B_\beta)$, is the average number of hashes computed per coin issued in the chain.

The Hashcap-to-Coinage Ratio is a metric that is similar to the Difficulty-to-Reward Ratio, but defined for a chain of blocks rather than for a miner, as per Definition 8. Using the previously defined terms for hashcap and coinage, we formulate $hcr(B_\beta)$ in Eq. 4.

$$hcr(b) = \frac{hashcap(b)}{coinage(b)} \quad (4)$$

This ratio is a key metric, as it enables the estimation of the value of a single cryptocurrency coin, as we will present in Sect. 3.3.

3.3 Coin Metrics

Insofar, in the previous two sections, we have presented basic metrics of interest related to mining throughput, difficulty, and rewards. In this section, we will utilize these metrics to construct valuation metrics which can be used to quantify the hash-based value gained, or lost, from mining coins.

Definition 9 (*Fungibility Dilution Factor* $fdf(m_\mu, B_\beta)$) The Fungibility Dilution Factor for a miner in a blockchain, denoted by $fdf(m_\mu, B_\beta)$, is the amount by which the hash-based cost of a miner's reward will be amplified in the blockchain.

We quantify how much gain, or loss, in hash-based value is made by a miner due to fungibility using the Fungibility Dilution Factor, presented in Definition 9. Using our previously defined metrics, dividing the miner's difficulty-to-reward ratio by the

blockchain's hashcap-to-coinage ratio results in the fungibility dilution factor, as per Eq. 5.

$$fdf(m_\mu, B_\beta) = \frac{drr(m_\mu)}{hcr(B_\beta)} \quad (5)$$

Coin fungibility dictates that all coins in a chain's coinage are perfectly interchangeable with one another. Because of this, once a new set of coins are created by a miner m_μ in a blockchain B_β , we treat them in our computationally-grounded framework as having a hash-based value equal to the chain's hashcap-to-coinage ratio, even if the difficulty-to-reward ratio that was exhibited by the miner in creating these coins was different (i.e. $hcr(B_\beta) \neq drr(m_\mu)$).

For a given miner and blockchain, when $fdf(m_\mu, B_\beta) > 1$, then the miner will receive a set of coins which represent a larger number of hashes than the miner performed to receive them (i.e. $hcr(B_\beta) > drr(m_\mu)$). On the other hand, when $fdf(m_\mu, B_\beta) < 1$, then the miner will receive a set of coins which represent a smaller number of hashes than performed (i.e. $hcr(B_\beta) < drr(m_\mu)$). When $fdf(m_\mu, B_\beta) = 1$, m_μ is issued in B_β a set of coins worth as many hash computations as were performed to receive them.

Consequently, we consider miners to have made a hash-based gain in our computationally-grounded framework if $hcr(B_\beta) > drr(m_\mu)$. In other words, if a miner is rewarded with a set of coins at a hash-based cost that is less than the chain's hashcap-to-coinage ratio, it was received a set of coins at a discount. This, of course, does not necessarily mean that the miner can make a profit selling its coins in a real-world market, where prices may not necessarily be dictated by hash-based valuation metrics. Similarly, a loss is said to have been incurred in our computationally-grounded framework, if $hcr(B_\beta) < drr(m_\mu)$.

Definition 10 (*Hash-Restitution Time* $hrt(m_\mu, B_\beta)$) The Hash-Restitution Time of a miner in a blockchain, denoted by $hrt(m_\mu, B_\beta)$, is the amount of time (in blocks) between m_μ first receiving a $reward(m_\mu)$, and the first block during which $hcr(B_\beta) \geq drr(m_\mu)$ holds true.

For the case when $fdf(m_\mu, B_\beta) < 1$, we introduce the Hash-Restitution Time in Definition 10, which is a means to analyse the amount of time a miner had to wait for the hashcap-to-coinage ratio to first meet the difficulty-to-reward ratio at which it created coins, i.e. the amount of time m_μ has to wait to be paid back in hash-based value.

Unlike previous metrics, $hrt(m_\mu, B_\beta)$ may not be immediately measurable for a miner that receives a reward while $fdf(m_\mu, B_\beta) < 1$. This is because it may not be easily determinable when exactly in the future $hcr(B_\beta)$ is expected to rise, due to instability in mining power or reward-scheme intrinsic reasons. In fact, for some PoW schemes, miners may never even be fully paid back if $hcr(B_\beta)$ never follows a sufficiently long upwards trend.

However, once this value is known, it serves as an indicator in our framework for m_μ to quantify the opportunity cost it had to pay before its rewarded coins attained

a hash-based value that is equal to the hash-based cost m_μ paid for them. In some cases, m_μ may never wait long enough, and transfer its coins.

Again, this metric is agnostic about both the specifics of the PoW blockchain it is used on, and the coin-spending behavior of miners. Its sole purpose in our computationally-grounded framework is to provide a measurement point that can be used to gain insight about how a reward-scheme delays fully compensating miners in hash-based value.

4 Inequity in Bitcoin

In this section, we apply our computationally-grounded framework introduced in Sect. 3 to Bitcoin, the most prominent realization of Nakamoto, practically analysing and quantifying the state of hash-based coin valuation in its network. Our analysis focuses on identifying and highlighting sources of inequity in Nakamoto, and demonstrating how these sources have affected Bitcoin’s miners and coinage in practice. To aid the reader in this section, we summarize our definitions from Sect. 3 in Table 1.

Table 1 Computationally-grounded framework definition and notation summary

Definition	Notation	Summary
1. Miner	m_μ	Uniquely identified by μ , has hashing throughput $power(m_\mu)$, and rewarded after $difficulty(m_\mu)$ hashes
2. Hash-Time-to-Issuance	$hti(m_\mu)$	Average amount of time before a hash computation is compensated
3. Reward	$reward(m_\mu)$	Expected number of coins received by m_μ as a reward for $difficulty(m_\mu)$ hashes on average
4. Difficulty-to-Reward Ratio	$drr(m_\mu)$	Average cost of a single coin in hashes for a miner m_μ
5. Blockchain	B_β	Uniquely identified by β
6. Hashcap	$hashcap(B_\beta)$	Expected number of hashes required to create B_β
7. Coinage	$coinage(B_\beta)$	Total number of reward coins issued in B_β
8. Hashcap-to-Coinage Ratio	$hcr(B_\beta)$	Average hash cost of coins in B_β
9. Fungibility Dilution Factor	$fdf(m_\mu, B_\beta)$	The ratio between the hash cost of a coin for m_μ , and the hash cost of coins in B_β
10. Hash-Restitution Time	$hrt(m_\mu, B_\beta)$	The amount of time before $fdf(m_\mu, B_\beta) \geq 1$ holds true

Bitcoin Adaptation. To model Bitcoin using our computationally-grounded framework, we integrate the Bitcoin-specific policies on difficulty and reward using the three following rules:

1. Only a single blockchain B_β is assumed to be mined on without forks.
2. For any miner m_μ , $difficulty(m_\mu)$ is equal to the current block mining difficulty for the Bitcoin blockchain B_β .
3. For any miner m_μ , $reward(m_\mu)$ is equal to the current Bitcoin block reward per its reward halving schedule.

The first rule applied to our framework is a simplifying assumption to focus the analysis on the best case scenario where consensus is working as intended. While this prevents the examination of the state of affairs during forks, it will highlight how even ideal conditions fail to establish equity amongst miners.

The second rule enables our framework’s definition of difficulty to follow that of Nakamoto, whereby mining a block is the only directly rewardable action in the protocol. We define $difficulty(m_\mu)$ to abstract away the out of scope details of Bitcoin’s difficulty adjustment algorithm, since they hold no repercussions for our analysis.

Similarly, the third rule applies Bitcoin’s reward schedule, which begins with 5, 000, 000, 000 coins (satoshis) and halves the reward every 210, 000 blocks.

4.1 Inequitable Hash-Time-to-Issuance

The fact that a miner has to wait for the chance to be a block’s creator in Nakamoto creates different hash-time-to-issuance costs for miners of relatively different sizes. This difference is presented in Fig. 1, which shows the linear relationship between a miner’s m_μ relative size and its expected $hti(m_\mu)$ value in Nakamoto. While other literature [10] analyzes this difference using the coefficients of variation of rewards received per block for a miner, the $hti(m_\mu)$ metric presented here gives a more concrete sense of the differences in time-related *opportunity costs* for miners of different relative sizes. These results extend directly to Bitcoin.

The cost implications of the aforementioned relationship in Nakamoto can be further elaborated by comparing two data points from the above plot. For example, consider two miners, one with 1% of the total network hashing power, and one with 0.01%. The first miner would have to experience an average delay of 200 blocks before receiving compensation for each hash it computes, while the second miner’s delay is 20, 000 blocks per hash. Under Bitcoin’s 10-minute average block interval,

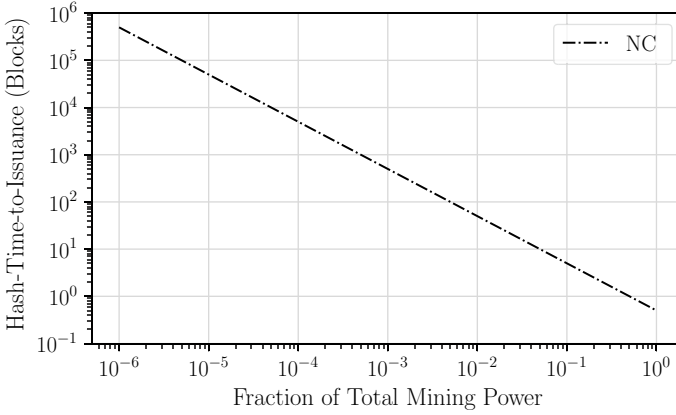
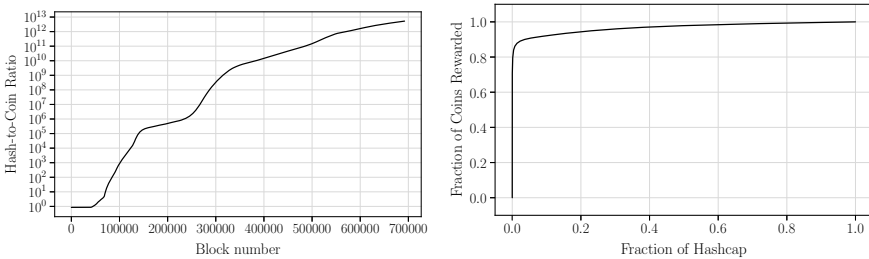


Fig. 1 Hash-Time-to-Issuance versus relative mining power in Bitcoin

the relatively larger miner can spend its mining rewards approximately every 1.39 days on average, while the smaller miner can only do so every 4.64 months approximately. This severely disproportional difference in reward delay between miners is not directly apparent from analyses which utilize the coefficients of variation.

4.2 An Increasing Hashcap-to-Coinage Ratio

Because Bitcoin’s reward scheme does not adjust the number of coins issued in response to the difficulty of mining, and mining difficulty has increased significantly since Bitcoin’s deployment, the Hashcap-to-Coinage Ratio has increased exponentially several times. Even with a stable mining difficulty, the reward-halving schedule causes the Hashcap-to-Coinage Ratio to increase. In Fig. 2, both the Hashcap-to-Coinage Ratio (Fig. 2a), and the distribution of the hashcap (Fig. 2b) over coins are



(a) Hash-to-Coin Ratio versus block number in Bitcoin.

(b) Cumulative Hashcap distribution over coin supply in Bitcoin.

Fig. 2 Hashcap-to-Coinage Ratio (HCR) plots

presented. The data presented in Fig. 2 was estimated using Bitcoin’s block difficulty parameter.

From a hash-based cost perspective, these increases have caused the running average hash-based cost of production for a single coin to increase dramatically over time in Bitcoin. The insight Fig. 2a offers is that almost all of Bitcoin’s hash-based value was computed since after 400, 000 blocks were created.

Because of Bitcoin’s reward scheme, which is adapted to this computationally-grounded framework using rules #2 and #3, the difficulty-to-reward ratio of all miners has only worsened over time as mining power has increased. To further understand the advantage earlier miners had, one can see how much hashing power was exchanged for different portions of the coin supply in Fig. 2b. Remarkably, the coin supply of Bitcoin has an incredibly skewed distribution, where more than 80% of the total coin supply was rewarded in exchange for less than 1% of the total number of hashes calculated to maintain the system.

4.3 Subsidy Through the Fungibility Dilution Factor

The inequity between miners which participate at different times is not just restricted to coin creation costs, but also extends to the hash-based value of received rewards. Because coins are perfectly interchangeable in the ledger, they are valued equally, even if some have higher hash-based costs than others. To quantify the extent to which this affects the hash-based valuation of mining rewards in Bitcoin, we present the Fungibility Dilution Factor of coins at the time of issuance in Fig. 3a, and across the coin supply in Fig. 3b.

If the FDF lies around a value of $1 \pm \epsilon$, where ϵ is some negligible value, then the hash-based value of the mining rewards can be considered to correspond to the number of hash computations performed to attain them. Instead, in Fig. 3a, it can be seen that Bitcoin’s FDF, after running at an almost constant value of 1 for a few

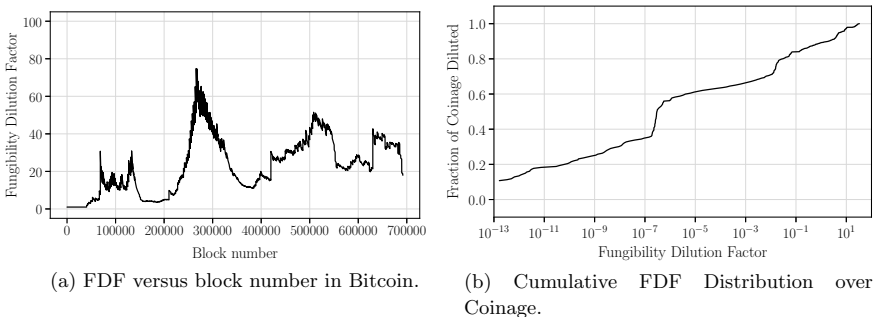


Fig. 3 Fungibility-Dilution Factor (FDF) plots

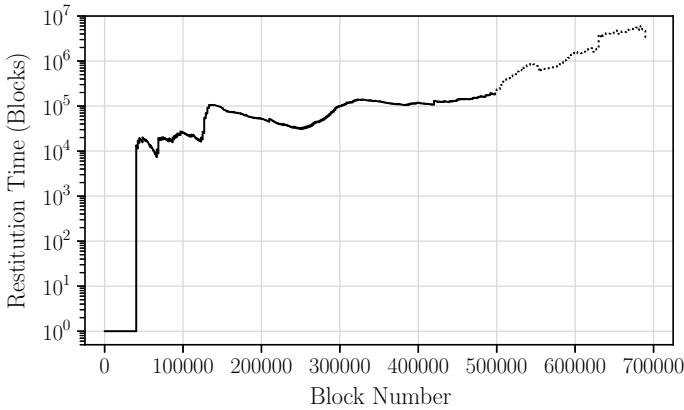


Fig. 4 Hashcap-Restitution time, in blocks, for each block reward in Bitcoin

thousand blocks, oscillates dramatically over time, leading the issued mining rewards to have a hash-based value³ that is tens of times less than their hash-based cost.

However, due to the fluctuations of the FDF, the fungibility dilution factor of coins changes after their issuance. Namely, Fig. 3b presents the cumulative distribution of the FDF over the coin supply so far. Remarkably, while the dilution factor has remained below 100 so far for Bitcoin, it has caused over 70% of the coinage to have its hashcap amplified by over 100-fold in value, with nearly 50% even growing by at least 1-million fold.

4.4 An Increasing Hash-Restitution Time

Given that Bitcoin’s fungibility dilution factor has remained well above 1 for most of its lifetime, the Hash-Restitution Time can be used to quantify how long an issued mining reward will take to reach a hash-based value that is at least equal to its hash-based cost.

In Fig. 4 we estimate the Hash-Restitution Time in blocks for the rewards of the first 500,000 Bitcoin blocks. As for the remaining blocks, since they have not yet reached an equitable hash-based value, their values are projected based on the assumption that the mining power remains the same as that of the last block used in the analysis.

Relatively shortly after the genesis block, Fig. 4 shows that miners had to wait for a number on the order of 10,000 blocks, or approximately 2-months, for the hash-based value of their rewards to reach their hash based costs. Starting from

³ At the time of reward issuance.

block 300,000, the Hash-Restitution Time goes up to the order of 100,000 blocks, or approximately 2 years.

Whether this is done by design to achieve a kind of lock-in effect, or simply an unintended consequence, the Hash-Restitution Time in Bitcoin seems to be uncontrolled, and trending towards impracticality for miners which participate relatively later in the protocol's lifetime.

Beyond Bitcoin While the analysis presented in this section only pertains to Bitcoin, the same methods and reasoning are applicable to other designs with minor adjustments. For example, to apply this approach to Ethereum, one would have to account for *uncle blocks*, factoring in their reward and mining difficulty into the chain's coinage and hashcap respectively.

5 Equitable Reward Constraints

Having defined our computationally-grounded framework, which we use to express equity in this paper, and demonstrated the current state of inequity in Bitcoin, we establish in this section a set of constraints for achieving equitable rewards in Proof-of-Work Mining. The main focus of the constraints in this section is to prevent miners with relatively small hashing powers, and miners which participate at relatively late, or early, stages of a blockchain's lifetime, from being disadvantaged.

Basic Approach. The main requirements that we use to establish the design constraints of this section are **proportionality**, and **timeliness**, of rewards, such that:

- The reward scheme issues coin rewards that have a hash-based value that is proportional to their miner's hash-based cost of attaining them.
- The reward scheme issues coin rewards in an amount of time bounded by the network's block creation interval.

The reasoning behind these requirements is to prevent any underpayment or devaluation in terms of hash-based value, allowing miners to receive coins with a hash-based value equal to their contribution to the system, without unjustifiable delays in compensation.

In Sects. 5.1 and 5.2, we motivate and present the design constraints in terms of the computationally-grounded framework from Sect. 3, such that each constraint is expressed as a set of restrictions that the protocol must place on the relevant framework metrics.

However, to achieve the notions of equity that we aim for, we propose a unique coin issuance approach. Namely, it can no longer be the case that a constant number of coins are issued per block in a winner-takes-all fashion, while having block mining difficulty be a variable. Instead, in our approach, we propose that mining rewards become uncapped. In Sect. 5.3, we examine the effects of adopting our unrestricted

reward scheme on coin-supply growth, and outline the conditions for relative coin supply stability.

5.1 *Undiluted Reward Constraints*

The first constraint, expressed in Eq. 6, aims to maintain a fixed correspondence between the number of hashes performed by miners, and the number of coins they receive in return.

$$hcr(B_\beta) \approx drr(m_\mu) \quad (6)$$

Such a constraint means that, unlike Bitcoin, for any miner to receive a coin, it must perform approximately $hcr(B_\beta)$ hash calculations. Essentially, this correspondence aims to stabilize the Difficulty-to-Reward Ratio of miners using the Hashcap-to-Coinage Ratio of the entire blockchain, such that miner rewards are not diluted.

Stabilizing this correspondence is an endeavor that is in stark contrast to existing blockchain reward schemes, which employ mechanisms that directly destabilize this relationship. For example, Bitcoin's so-called *halving* schedule fundamentally causes spikes in Bitcoin's HCR. These spikes are further exacerbated by changes in Bitcoin's block mining difficulty, which does not affect how many coins are rewarded per block. Whether Bitcoin's HCR instability is by design, or beneficial, our approach is to keep the Hashcap-to-Coinage ratio relatively stable, and avoid directly granting any advantages to miners which participate during periods of lower block mining difficulty.

Remarkably, when the HCR is unstable, the fungibility of coins distorts the hash-based cost of production for the entire coin supply. In Bitcoin, this leads to a situation where a majority of its coin supply is produced at a marginal cost compared to the remainder, yet the entire coinage is treated as having equal face-value. Under our approach on the other hand, the goal is to not dilute miner rewards, and stabilize the Fungibility Dilution Factor for all miners of the chain as expressed in Eq. 7.

$$fdf(m_\mu, B_\beta) \approx 1 \quad (7)$$

Of course, these constraints do not cover cases where miners willingly forfeit rewards, or receive a penalty for misbehavior in consensus. In such circumstances, the hashes computed by such miners will go unrewarded, theoretically increasing the HCR of the blockchain.

5.2 *Prompt Restitution Constraints*

Our second reward constraint is the prompt, and direct, distribution of coins to miners in exchange for their hash calculations. This is in contrast to distributing rewards

indirectly on the long run using lotteries, or any other mechanisms similar to the round based winner-takes-all Bitcoin dynamic.

More concretely, each miner is rewarded on average at least once per block for each of its search attempts for a PoW, instead of having to wait for some expected number of blocks to be created before being credited, following the constraint described by Eq. 8.

$$hti(m_\mu) \approx \text{blocktime}(B_\beta) \quad (8)$$

For example, a Bitcoin miner with one third of the total network mining power would have to wait two blocks on average before receiving its block reward. On the other hand, an equitably treated miner with the same hashing power would expect to receive a third of the total coins rewarded to the network every block.

This constraint, in conjunction with that of Sect. 5.1, lead towards establishing a short and stable Hash-Restitution Time for all miners, as presented in Eq. 9.

$$hrt(m_\mu, B_\beta) \approx 1 \quad (9)$$

In stark contrast, as demonstrated in Sect. 4, Bitcoin miners have no guarantees on how many blocks they would have to wait before the hash-based value of their reward coins makes up for the hashing power they expended to create them.

5.3 *Equitable Coin Supply Growth*

Achieving equitable rewards in this fashion means that miners accumulate rewards that are proportional to their mining power, leaving the number of new coins that can be created at any given moment virtually uncapped. Practically, however, the total amount of computational power invested by miners in the network restricts the growth of the supply of coins, preventing the relative growth rate of the coin supply from spiraling out of control. More precisely, over time, the relative growth rate of the coin supply tends towards the relative growth rate of mining power over time. Consequently, as long as the amount of computational power invested in mining remains stable, the relative growth rate of coin supply slowly converges to zero. Under such constant mining power, the coin supply grows only by a constant number of coins per block similar to Bitcoin's, converging towards a relative growth rate of 0. Linearly growing mining power leads to the same convergence, but at a slower rate.

In Fig. 5 we illustrate coin supply growth under different mining power growth rates. To plot this figure, we simulated a simple proportional reward issuance scenario under three different mining difficulty settings. In the first setting, the amount of mining power stays constant, while in the second, the mining power grows by one unit every time period. In the third setting, the mining power is multiplied by 1.025 each time period, denoting a 2.5% increase per period. As we start from a coin supply of zero, the initial relative growth in all scenarios is substantial. However, as time passes, the relative rates of increase in coin supply and mining power converge.

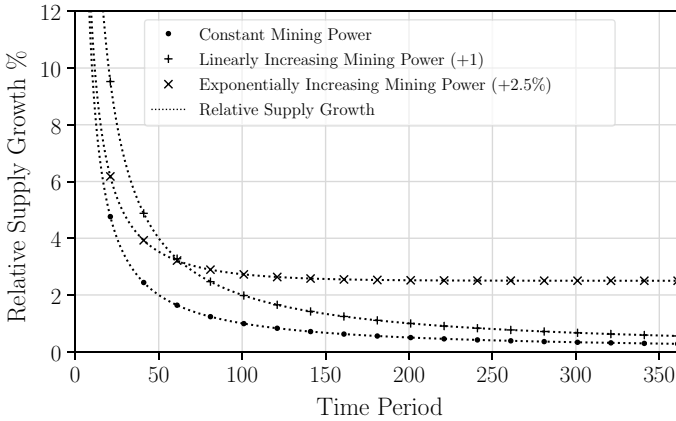


Fig. 5 Plot of Time versus Relative Coin Supply Growth (in Percentage) under three different mining power growth scenarios. Under constant mining power over all time periods (\bullet) the relative supply growth per time period of coins goes to zero over time. Similarly, under linear growth of mining power ($+$), where 1 more unit of power is added per time period, the relative supply growth per time period also goes to zero over time. However, under exponential mining power growth (\times), where 2.5% more power is added per time period, the relative supply growth converges to the relative mining power growth over time

6 Conclusion

In this work, we introduced a computationally-grounded framework for quantifying the relationship between miner expenditures and compensation. In this framework, we used hashing power as an objective cost basis for coin creation, and used newly minted coins as an objective measure of compensation, while intentionally avoiding external variables, such as hardware costs or currency exchange rates. Furthermore, we used the aforementioned computationally-grounded framework to demonstrate the inequity in state-of-the-art PoW mining reward schemes using Bitcoin as an example. We empirically demonstrated the effects of winner-takes-all lottery dynamics on Bitcoin, showing them to have caused a large discrepancy between miner rewards for relatively small and late miners. Small miners were shown to have to wait much longer on average than larger miners to receive any compensation, while miners who participate during late periods of relatively more expensive coin creation costs were shown to be at several disadvantages. Lastly, we introduced constraints for achieving equitable rewards, arguing that stability of the hashcap-to-coinage ratio is akin to equity of rewards issued to miners who participate at different times, and that the stability of the hash-time-to-issuance metric, which requires a more continuous reward issuance schedule for miners of all sizes, enforces equity in compensation delays. Furthermore, we have shown that when adhering to such constraints, coin supply growth dynamics exhibit reasonably similar behaviors to those of constant rewards over time, such that relative coin supply growth becomes proportional over time to relative mining power growth.

Future Work. By applying our framework to cryptocurrencies other than Bitcoin, we will analyze and compare the mining reward equity of different coins. To further simplify this comparison, we will introduce new scoring metrics based on our computationally grounded framework to enable the direct numeric comparison of mining reward equity between different blockchains.

Acknowledgements We'd like to thank our anonymous reviewers for their valuable feedback on this paper. This work was made possible with the generous funding and support of the Imperial College London President's PhD Scholarship program.

References

1. Bissias, G., Levine, B. N. (2020). Bobtail: Improved blockchain security with low-variance mining. In *ISOC Network and Distributed System Security Symposium*
2. Chen, X., Papadimitriou, C., & Roughgarden, T. (2019). An axiomatic approach to block rewards. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (pp. 124–131).
3. Dong, Y., & Boutaba, R. (2019). Elasticoin: Low-volatility cryptocurrency with proofs of sequential work. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 205–209). IEEE.
4. Dong, Y., & Boutaba, R. (2020). Melmint: Trustless stable cryptocurrency. *Cryptoeconomic Systems*.
5. Kwon, Y., Liu, J., Kim, M., Song, D., & Kim, Y. (2019). Impossibility of full decentralization in permissionless blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (pp. 110–123).
6. Nakamoto, S., et al. (2008). Bitcoin: A peer-to-peer electronic cash system.
7. Orlicki, J. I. (2020). Sequential proof-of-work for fair staking and distributed randomness beacons. arXiv preprint [arXiv:2008.10189](https://arxiv.org/abs/2008.10189).
8. Pass, R., & Shi, E. (2017). Fruitchains: A fair blockchain. In *Proceedings of the ACM Symposium on Principles of Distributed Computing* (pp. 315–324).
9. Sompolinsky, Y., & Zohar, A. (2015). Secure high-rate transaction processing in bitcoin. In *International Conference on Financial Cryptography and Data Security* (pp. 507–527). Springer.
10. Szalachowski, P., Reijsbergen, D., Homoliak, I., & Sun, S. (2019). Strongchain: Transparent and collaborative proof-of-work consensus. In *28th USENIX Security Symposium (USENIX Security 19)* (pp. 819–836).
11. Trzeszczkowski, K. Ergon—stable peer to peer electronic cash system, <https://ergon.moe/>
12. Trzeszczkowski, K. (2021). Proportional block reward as a price stabilization mechanism for peer-to-peer electronic cash system.

Market Equilibria and Risk Diversification in Blockchain Mining Economies



Yun Kuen Cheung, Stefanos Leonardos, Shyam Sridhar,
and Georgios Piliouras

Abstract The success of blockchain-based applications, most notably cryptocurrencies, has brought the allocation of mining resources at the epicenter of academic and entrepreneurial attention. Critical for the stability of these markets is the question of how miners should adjust their allocations over time in response to changes in their environment and in other miners' strategies. In this paper, we present a *proportional response* (PR) protocol that makes these adjustments for any risk profile of a miner. The protocol has low informational requirements and is particularly suitable for such distributed settings. When the environment is static, we formally show that the PR protocol attains stability by converging to the market equilibrium. For dynamic environments, we carry out an empirical study with actual data from four popular cryptocurrencies. We find that running the PR protocol with higher risk diversification is beneficial both to the market by curbing volatile re-allocations (and, thus, increasing market stability), and to individual miners by improving their profits after accounting for factor mobility (switching) costs.

Keywords Blockchain mining · Market equilibria · Proportional response dynamics · Risk diversification · Factor mobility · Cryptocurrencies

Y. K. Cheung
Royal Holloway, University of London, Egham, United Kingdom
e-mail: yunkuen.cheung@rhul.ac.uk

S. Leonardos (✉)
King's College London, London, United Kingdom
e-mail: stefanos.leonardos@kcl.ac.uk

S. Sridhar · G. Piliouras
Singapore University of Technology and Design, Somapah, Singapore
e-mail: shyam.sridhar@ethereum.org

G. Piliouras
e-mail: georgios@sutd.edu.sg

S. Sridhar
Ethereum Foundation, Bern, Switzerland

1 Introduction

The massive growth of permissionless blockchains has led to the development of numerous disruptive technologies, including decentralized applications (DApps) and finance (DeFi) protocols [70, 74], smart contracts [13] and most notably, cryptocurrencies [63]. Critical actors in the evolution of the blockchain ecosystem are the *miners* who provide costly resources, e.g., computational power in Proof of Work (PoW) [45, 58] or monetary tokens of the native cryptocurrency in Proof of Stake (PoS) protocols [18, 47, 57], to secure the reliability of the supported applications. For their service, miners receive monetary rewards in the form of transaction fees and newly minted coins, typically in proportion to their allocated resources [14, 20]. Yet, miners act in self-interested ways, and may enter, leave, or switch mining networks at any given time. If their incentives are not aligned with the common interest, this behavior can cause unexpected (and undesirable) fluctuations in mining supplies, which in turn, can critically undermine the ecosystem’s goals for growth and security [30, 46]. This has brought miners’ behavior under the spotlight of academic and managerial research [3, 12, 43, 72]. However, due to their inherent complexity, mining networks pose novel challenges that cannot be adequately addressed by conventional game-theoretic and economic models. The reasons are manifold.

First, as networks continue to grow in both size and numbers, individual miners forfeit *market power*, i.e., influence on aggregate outcomes (e.g., cryptocurrency prices as in an oligopoly [2]), and early stage, small-scale, game-theoretic interactions give place to *large market, multi-agent systems* in which network metrics (e.g., aggregate resources and prices) are perceived as exogenous signals [32, 44]. Second, miners’ decisions need to account for the constantly changing environment and other miners’ actions and are, thus, highly dynamic in nature. Thus, if not coupled with proper algorithmic approaches, static game-theoretic or market models offer only limited views on miner’s incentives. Even worse, popular online optimization dynamics with good *on average* but unpredictable day-to-day performance (e.g., Multiplicative Weights Updates [4, 26–29, 65]) become essentially useless in this setting.

Additional issues in the study of miners’ incentives involve *factor mobility* costs and miners’ inhomogeneous *risk profiles*. The former concern all costs incurred by re-allocations of mining resources. The increasing popularity of virtual mining [5], secondary (resale or rental) markets of mining gear [7] and proxy schemes (e.g., delegated PoS [76]) to enter or leave mining enable such re-allocations between blockchains using not only compatible (e.g., fork networks like Bitcoin and Bitcoin Cash), but also incompatible technologies [49]. The latter concern miners’ different *risk diversification* profiles, ranging from risk aversion to risk seeking, and the different strategies that this profiles dictate to hedge against the extreme swings in cryptocurrency valuations.

Model and Contributions. Motivated by the above, we set out to model mining networks and reason about miners’ behavior. We assume that miners perceive market

metrics (aggregate resources, mining revenues and cryptocurrency prices) as exogenous, independent signals. Each miner is endowed with a finite capacity of resources and maintains their own mining cost (for each available blockchain) and individual risk preferences. In mathematical terms, the resulting *blockchain mining economy* mirrors a Fisher market with *quasi Constant Elasticity of Substitution* (quasi-CES) utilities and substitution parameters, ρ , in $(0, 1]$ (the less the ρ , the more risk averse a miner is). Our research goals involve the search of a dynamic allocation rule to learn miners' equilibrium allocations under arbitrary market conditions, and the empirical analysis of miners' equilibrium behavior. Our contributions are the following.

Concerning our first goal, we derive a *proportional response* (PR) protocol which converges *globally*, i.e., for any combination of miners' risk profiles, and positive initial allocations and any mining revenues, to the unique market equilibrium of the described blockchain mining economy. The PR dynamics have been shown to achieve remarkable stability in different multi-agent settings [6, 9, 23–25, 44, 75, 77] and express the following natural idea: in each round, the miner allocates their resources in proportion to the utilities that they obtained from the various mining etworks in the previous round. To address the challenges described above, we consider the newly proposed economy of *Fisher markets with quasi-CES utilities*, and derive a novel version of the PR protocol. The PR protocol requires as inputs only miner-dependent information (mining costs, risk profile, and budget) and network-dependent information (allocated resources and generated revenues) that can be easily observed or estimated. Thus, it provides an *individual miner's* perspective against real data which is useful to both individual miners for updating their day-to-day allocations and to researchers studying miners' behavior.

In the context of our second goal, we then turn to study miners' allocations using the PR protocol in an empirical setting with daily data from four popular cryptocurrencies (coins): Bitcoin, Bitcoin Cash, Ethereum and Litecoin. Our results provide insights on the effects of miners' risk profiles and factor mobility costs on equilibrium allocations. Unless miners are fully risk averse (in which case, they uniformly distribute their resources on the available coins), a latent driver of their decisions is the normalized *efficiency ratio* (aggregate revenues over estimated expenses) of the considered cryptocurrencies. As risk diversification decreases, miners deviate from the uniform distribution to distributions that trace the efficiency ratio. At $\rho = 1$ (lowest risk diversification), miners allocate all their resources to the coin with the highest efficiency ratio. This results in frequent and dramatic re-allocations of resources between different coins and undermines system stability.

While, in the naive approach, individual profits are maximized under minimal risk diversification, the figures tell a different story when we account for factor mobility costs. Even if such costs are relatively low, they dramatically reduce the effectiveness of risk seeking strategies (that dictate frequent re-allocations to the best performing coins) and profit maximization obtains for parameter values in the interior of the $(0, 1)$ interval, i.e., for risk profiles strictly between the extremes of risk aversion and risk seeking. Thus, by contributing to both miner's individual profitability and to the stability of the mining networks, this provides the first formal evidence that risk diversification, along with market size, and restrictions in factor mobility (as e.g.,

enforced by the technological polyglossia of blockchain networks) can be instrumental in aligning miner’s incentives with the ecosystem’s long-term goals.

Other Related Works. Our paper contributes to the growing literature on blockchain economies. Academic research [8, 38, 72] and investors’ sentiment [60, 73], both provide ample evidence that miners’ behavior is an understudied area, still at its infancy [1]. However, interest on miners’ incentives comes from many different directions, including environmental concerns [33, 34, 71], decentralization of mining networks [2, 53, 54], in-protocol adversarial behavior [40, 50, 69], protocol instabilities and design [11, 15, 19, 36] and economic analyses [64] among others. Existing studies mostly shed light on allocations in *single* blockchains [42, 48] or small-scale interactions, e.g., between two mining pools [1], and raise questions regarding miners’ incentives in larger settings. Our paper makes a step forward precisely in this direction and shows (among others) that many small-world phenomena (e.g., concentration of power [2] or instabilities of common learning rules [25]) tend to fade as mining networks grow in size and diversity. In this direction, most closely related is the work of [72]. Technically, our setting mirrors the abstract model of a Fisher market with quasi-CES utilities and extends recent developments in the setting of distributed production economies [10, 25]. Our new version of PR protocol might be of independent interest in market dynamics [9, 21, 22, 37].

2 Model: Mining Economies

We consider a setting with $N = \{1, 2, \dots, n\}$ miners and $M = \{1, 2, \dots, m\}$ *mineable* cryptocurrencies (or coins).¹ For $k \in M$, let v_k denote the aggregate revenue, e.g., newly minted coins and transaction fees, generated by cryptocurrency k and assume that v_k is distributed to miners proportionally to their allocated resources. For $i \in N$ and $k \in M$, let c_{ik} be miner i ’s cost to allocate one unit of resource to the mining of cryptocurrency k . Using standard conventions, we will write $X_k := \sum_{j \in N} x_{jk}$ for the aggregate resources $\mathbf{x} = (\mathbf{x}_i, \mathbf{x}_{-i})$ for the vector of allocated resources in cryptocurrency $k \in M$, where $\mathbf{x}_i = (x_{ik})_{k \in M}$, and $\mathbf{x}_{-i} = (\mathbf{x}_{-ik})_{k \in M}$ are the allocations of miner i and of all miners other than i , respectively.

Quasi-CES Utilities Based on the above, the utility of a miner $i \in N$ from mining a single cryptocurrency k takes the form $v_k \cdot x_{ik} / X_k - c_{ik}x_{ik}$. When considering multiple cryptocurrencies, rather than using physical resources, it will be convenient to express all allocations in common monetary units (e.g., US dollars). Thus, henceforth, we will assume that each miner $i \in N$ is endowed with a finite monetary capacity, $K_i > 0$, of resources and we will write $b_{ik} := c_{ik}x_{ik}$ to denote the *spending* of miner i at mining cryptocurrency $k \in M$. Thus, a strategy of miner i will be described by a non-negative vector $\mathbf{b}_i = (b_{ik})_{k \in M}$ which satisfies $\sum_{k \in M} b_{ik} \leq K_i$.

¹ Here, mining refers to any form of expense or investment of scarce resources such as computational power in PoW or native tokens in PoS.

As above, we will write $\mathbf{b} = (\mathbf{b}_i, \mathbf{b}_{-i})$ to denote the spending profile of miner i and all miners other than i , respectively, and $b_k = \sum_{j \in N} b_{jk}$ to denote the aggregate spending in mining cryptocurrency $k \in M$. Using this notation, the utility of a miner $i \in N$ from mining cryptocurrency k is expressed as $v_{ik}b_{ik} - b_{ik}$, where $v_{ik} := v_k/(c_{ik}X_k)$ denotes the *estimated efficiency ratio* of cryptocurrency k as calculated by miner i , i.e., revenues over expenses where the expenses are estimated using miner i 's cost. For the interpretation of our empirical results, it will also be useful to define the *normalized efficiency ratio* or simply *efficiency ratio*, e_{ik} , of cryptocurrency k as estimated by miner i

$$e_{ik} := v_{ik} / \sum_{l \in M} v_{il}, \quad \text{for all } k \in M. \quad (1)$$

A naive generalization of the above utility to multiple cryptocurrencies results in the following expression

$$u_i(\mathbf{b}_i, \mathbf{b}_{-i}) = \sum_{k=1}^m (v_{ik}b_{ik} - b_{ik}) \quad (2)$$

However, this aggregation ignores the risk diversification profile of the miner and the degree of factor (resource) mobility between various cryptocurrencies. To address these considerations, we introduce *diminishing marginal utility* of mining revenues which is modeled via concave utility functions of the form $u_i(x) = x^{\rho_i}$, with $0 < \rho_i \leq 1$, for each miner $i \in N$. When aggregated over all cryptocurrencies $k \in M$, they amount to a *quasi Constant Elasticity of Substitution (quasi-CES)* utility function

$$u_i(\mathbf{b}_i, \mathbf{b}_{-i}) = \left(\sum_{k=1}^m (v_{ik}b_{ik})^{\rho_i} \right)^{1/\rho_i} - \sum_{k=1}^m b_{ik}. \quad (3)$$

For $\rho_i = 1$, Eq. (3) reduces to the *quasi-linear* utility function of Eq. (2). Summing up, we will denote a *blockchain mining economy* defined by the utilities in Eq. (3) with $\Gamma = (N, M, (u_i, v_{ik}, \rho_i, K_i)_{i \in N})$. Whenever relevant, we will add the argument $t > 0$ to denote time-dependence in the above quantities.

Large Market Assumption. Observe that v_{ik} depends on X_k , which in turns depend on x_{ik} . This implies that miner i has *market power*. Specifically, when x_{ik} appears in the aggregate market resources, X_k , then, miner i takes into account the influence of their individual allocations on aggregate market metrics (total allocated resources and hence, cryptocurrency prices, total generated mining revenue etc.) in their strategic decision making. However, as mining networks grow larger, individual miners reckon cryptocurrency revenues as exogenously given (and not as variables that depend on their own actions). To formalize this notion, we make a *large market assumption* and assume that a vector of aggregate spending $\tilde{\mathbf{b}} = (\tilde{b}_k)_{k \in M}$, i.e., the total spending at each cryptocurrency, is exogenously given. This implies that the X_k 's and, hence, the v_{ik} 's do not depend on the allocation of miner i . Thus, given $\tilde{\mathbf{b}}$ each miner $i \in N$ selects an *optimal budget allocation* \mathbf{b}_i^* in $\arg \max_{\mathbf{b}_i} u_i(\mathbf{b}_i \mid \forall k \in M, X_k = \tilde{b}_k)$ which maximizes their utility subject to the constraint $\sum_k b_{ik} \leq K_i$.

Solution Concept. Together with the *static environment* assumption of v_k being fixed, and the *cost-homogeneity assumption* that for each $k \in M$, c_{ik} 's are the same for all $i \in N$,² the game-theoretic model of the blockchain mining economy, Γ , becomes mathematically equivalent to a *Fisher market model with quasi-CES utilities*. This approach is in line with [25, 31] who introduce this method to model and analyze large distributed production economies. The corresponding solution concept is the *market equilibrium* (ME).

Definition 1 (*Market Equilibrium (ME)*) Consider a blockchain mining economy, Γ . Then, a vector of aggregate allocation $\tilde{\mathbf{b}} = (\tilde{b}_k)_{k \in M}$ is a *market equilibrium (ME)* if there exists an optimal budget allocation $\mathbf{b}_j^* = (b_{jk}^*)_{k \in M}$ for each miner $j \in N$, so that $\sum_{j \in N} b_{jk}^* = \tilde{b}_k$ for each cryptocurrency $k \in M$. The vector $\mathbf{b}^* = (\mathbf{b}_j^*)_{j \in N}$ is called a *market equilibrium allocation* (or *spending*).

3 Proportional Response Dynamics

The Fisher market abstraction provides a handy way to determine the (unique) ME of Γ *algorithmically*, i.e., through a distributed *Proportional Response* protocol that converges to the ME of Γ for all possible combinations of miners' risk profiles and for any positive initial allocations (globally). To formulate the protocol, we first introduce some minimal additional notation. We write $r_{ik}(t) := (v_{ik} b_{ik}(t))^{\rho_i}$ and $r_i(t) := \sum_{k=1}^m r_{ik}(t)$ to denote miner i 's revenue (raised to the power of i 's parameter ρ_i) from cryptocurrency k and all cryptocurrencies, respectively, and $w_i(t) := K_i - \sum_{k=1}^m b_{ik}(t)$ to denote miner i 's unspent budget at time $t \geq 0$. We also define $\tilde{K}_i(t) := K_i \cdot (K_i - w_i(t))^{\rho_i - 1}$ for each $i \in N$.

Definition 2 (*Proportional Response Protocol (PR-QCES)*) Let Γ denote a blockchain economy with quasi-CES utilities. Then, the *Proportional Response (PR-QCES)* protocol is defined by

$$b_{ik}(t+1) := \frac{r_{ik}(t) \cdot K_i}{\max\{r_i(t), \tilde{K}_i(t)\}}, \quad \text{for } t \geq 0, \quad (\text{PR-QCES})$$

for any miner $i \in N$. An implementation of the (PR-QCES) protocol in pseudocode is provided in Algorithm 1.

Intuitively, the (PR-QCES) update rule suggests that if the revenue of miner i is high enough at round t , i.e., if $r_i \geq \tilde{K}_i$, then miner i will re-allocate all their resources in round $t+1$ in proportion to the generated revenues, r_{ik}/r_i , in round t . By contrast, if $r_i < \tilde{K}_i$,

² Since PoW mining resembles an oligopoly, conventional oligopolistic competition suggests that cost asymmetries cannot be very large among *active* miners [2, 55]. Similarly, in PoS, all miners (or validators) experience the same opportunity cost when committing their stake.

Algorithm 1 PR-QCES Protocol

Network input: network hashrate, X_k , and revenue, v_k , of each cryptocurrency $k \in M$.

Local input: miner i 's unit cost, c_{ik} , budget capacity, K_i , and risk parameter, ρ_i .

Output: equilibrium allocations, b_{ik} , $k \in M$, for each miner $i \in N$.

```

1: Initialize: spending (allocation)  $b_{ik} > 0$  for all  $k \in M$ .
2: loop over  $t \geq 0$  till stopping condition
3:   for each miner  $i \in N$  do
4:     procedure AUXILIARY( $(X_k, v_k, c_{ik})_{k \in M}, K_i, \rho_i$ )
5:        $v_{ik} \leftarrow v_k / X_k c_{ik}$ 
6:        $w_i \leftarrow K_i - \sum_{k \in M} b_{ik}$  ( $\triangleright$ ) unspent budget
7:        $\tilde{K}_i \leftarrow K_i (K_i - w_i)^{\rho_i - 1}$ 
8:        $r_{ik} \leftarrow (v_{ik} b_{ik})^{\rho_i}$  and  $r_i \leftarrow \sum_{k \in M} r_{ik}$ 
9:     procedure PR-DYNAMICS( $(r_{ik})_{k \in M}, r_i, K_i, \tilde{K}_i$ )
10:    if  $r_i > \tilde{K}_i$  then
11:       $b_{ik} \leftarrow r_{ik} K_i / r_i$ 
12:    else  $b_{ik} \leftarrow r_{ik} K_i / \tilde{K}_i$ 

```

then miner i will behave cautiously and allocate only a fraction of their resources in the next round. This fraction is precisely equal to the generated revenue at round t , i.e., $r_i(t)$, again in proportion to the revenue generated by each cryptocurrency $k \in M$.

For practical purposes, it is important that the (PR-QCES) protocol has low informational requirements. As can be seen from Algorithm 1, it uses as inputs only miner-specific (local) information (individual capacity and mining cost) and network-level information (aggregate revenue and hashrate) that can be either observed (revenue) or relatively accurately estimated (hashrate).

Convergence to ME The intuition behind the (PR-QCES) protocol is natural, except maybe for the particular choice of \tilde{K}_i . This choice is derived to ensure an important property: under the *static environment* and the *cost-homogeneity* assumptions if all miners use (PR-QCES), then it converges to the ME allocations of Γ for any selection of the $\rho_i \in (0, 1]$ and any strictly positive initial allocation, $b_{ik}(0) > 0$ for all $i \in N, k \in M$. This is formally stated in Theorem 1, our main theoretical result.

Theorem 1 (Equilibrium Mining Allocations) *Consider a blockchain economy Γ . Then, for any positive initial allocation (spending) vector, $\mathbf{b}^0 > 0$, the (PR-QCES) dynamics converge to the unique market equilibrium allocation, \mathbf{b}^* , of Γ .*

The proof of Theorem 1 is deferred to Appendix A.

4 Experiments

Having established convergence to ME allocations of the (PR-QCES) protocol, we now turn to study miners' behavior using empirical data. As argued above, the

(PR-QCES) protocol allows us to adopt an *individual miner's* perspective against exogenously given network metrics and obviates the need to generate synthetic data concerning (hypothetic) profiles of other miners. Importantly, implementing Algorithm 1 does not require knowledge of other miners' costs nor the assumption of cost-homogeneity.

4.1 Data Set and Experimental Setup

Our network-level data ("Network input" in Algorithm 1) has been sourced from coinmetrics.io and consists of daily observations of the (estimated) aggregate network hashrate, X_k , in Terahashes per day (TH/day) and aggregate miners' revenue, v_k , in USD (coinbase transactions and fees) for four PoW cryptocurrencies: Bitcoin (BTC), Bitcoin Cash (BCH), Ethereum (ETH) and Litecoin (LTC), in the period between January 01, 2018 and September 19, 2021 ($T = 1358$ days).³ The network-level data is visualized in Figs. 1 and 2. For our miner-level data ("Local input" in Algorithm 1), we only need to derive estimates for c_{ik} , i.e., for the cost of a miner to produce one unit of resource (one TH/s for a whole day) in each of the four cryptocurrency networks. Miner i 's risk profile, $\rho_i \in (0, 1]$ will be a *target parameter* that will vary through the experiments and miner i 's budget, K_i , will be assumed to be constant throughout the study period and normalized to 1 monetary unit (USD).

Mining Costs. To derive an estimate of a miner's cost, c_{ik} , to mine each cryptocurrency $k = 1, 2, 3, 4$, we collect data regarding electricity prices and prices of state-of-the-art mining equipment in each calendar year of the considered time period. The corresponding data is presented in Tables 1 and 2. The electricity cost estimates are based on the currently increasing trends (for 2021) and on the findings of [33, 34] for the 2018–2020 period, who suggest that kWh prices follow seasonal variations (due to weather dependent fluctuations in mining hotspots, e.g., China) and a constant to slightly decreasing overall trend between 2018 and 2020. Using the above, the cost, c , of a miner to produce one TH/s throughout a day to mine a given cryptocurrency is

$$c = \frac{P}{365 \cdot L_s \cdot H_s} + \frac{(W/1000) \cdot c_W \cdot 24}{H_s}, \quad (4)$$

where, as in Table 1, P denotes the mining equipment' acquisition price in USD, L_s its useful lifespan (assumed to be 2 years for all models), H_s its effective hashrate (in TH/s), W its power consumption (in Watt), and c_W the average cost per kWh in USD.

³ This period is before the implementation of the EIP-1559 update in the Ethereum transaction fee market [41, 56, 62, 66].

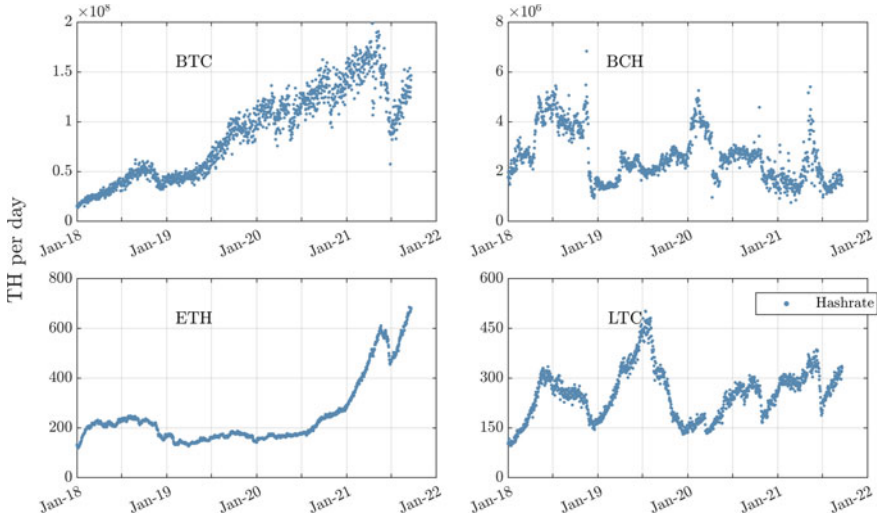


Fig. 1 Daily estimated hashrate, X_k , in TeraHashes per day for the four cryptocurrencies: Bitcoin (BTC), Bitcoin Cash (BCH), Ethereum (ETH) and Litecoin (LTC)

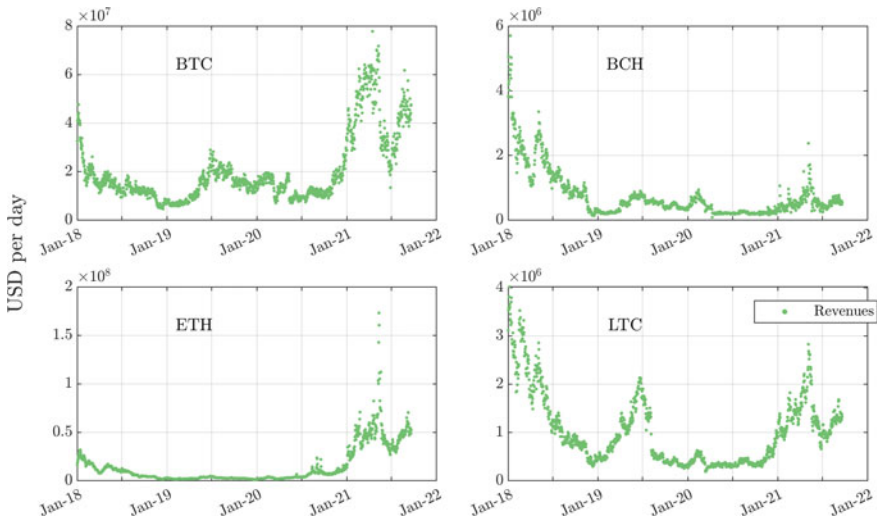


Fig. 2 Daily aggregate revenue, v_k , for the four cryptocurrencies: BTC, BCH, ETH and LTC. Source coinmetrics.io

Remark 1 (Data Reliability) Given the ambiguity concerning electricity prices faced by various miners and the variability of mining equipment prices among different sellers and time periods, in our complete set of simulations (not presented here), we have tested a wide range of values for the above parameters. While the equilib-

Table 1 Data used to estimate mining costs in Eq. (4). The lifespan, L_s , of all models is assumed to be 2 years

Coin	Year	Mining model	Price	Hashrate	Power (W)
BTC/BCH	2018	Antminer s9	\$1,900	14 TH/s	1372
	2019	Ebang Ebit E11+	\$1,400	37 TH/s	2035
	2020	Antminer s17+	\$2,080	73 TH/s	2920
	2021	Antminer s19 Pro	\$2,407	110 TH/s	3250
ETH	2018	PandaMiner B3 Pro	\$2,196	230 MH/s	1250
	2019	PandaMiner B3 Pro	\$2,034	230 MH/s	1250
	2020	PandaMiner B9	\$3,280	330 MH/s	1050
	2021	PandaMiner B7 Pro	\$2,684	360 MH/s	1650
LTC	2018	Antminer L3+	\$260	504 MH/s	800
	2019	Antminer L3++	\$320	596 MH/s	1050
	2020	Antminer L3++	\$430	596 MH/s	1050
	2021	Antminer L3+	\$237	504 MH/s	800

Sources asicminervalue.com and cryptocompare.com

Table 2 Average prices per kWh (on a global estimate). The figures are updated every six months (January-June and July-December)

Electricity costs (c_W)							
2018		2019		2020		2021	
\$0.070	\$0.080	\$0.065	\$0.075	\$0.065	\$0.060	\$0.065	\$0.070

Main sources [33, 34]

rium allocations may change, the qualitative findings remain throughout robust and equivalent to the ones presented here.

4.2 Empirical Results

Equilibrium Allocations Our first set of results concerns the effects of a miner's risk profile, $\rho_i \in (0, 1]$, on their equilibrium allocations. The results for four values of ρ_i are shown in the lower tiles of Fig. 3. The most important finding is that miners' allocations are driven by the *efficiency ratio*, e_{ik} , i.e., the normalized ratio between revenue and estimated (by miner i) expenses of each cryptocurrency k (cf. Eq. (1)).

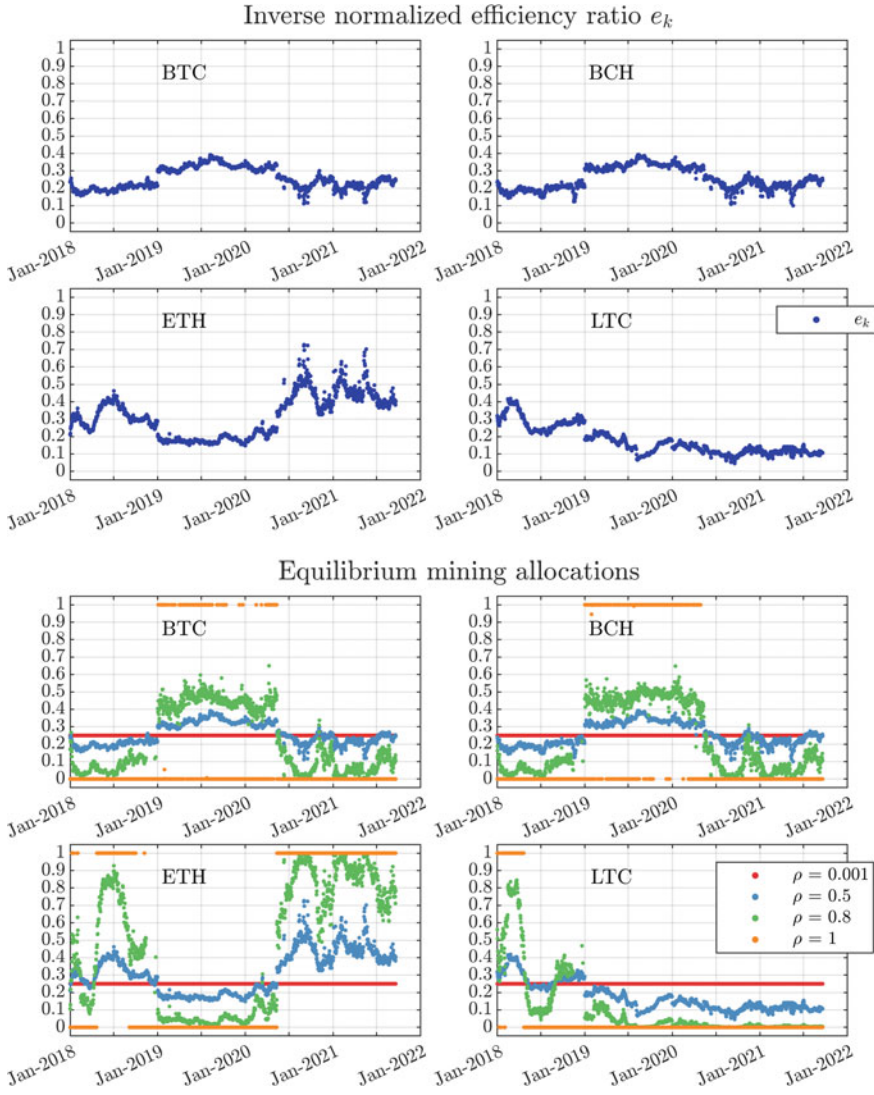


Fig. 3 Efficiency ratio (upper tiles) and equilibrium allocations for four different risk profiles ρ (lower tiles) in the mining economy of the case study. When ρ is close to zero (risk aversion), the miner uniformly distributes their resources among the available coins. As ρ increases, the allocations are driven by e_k : for $\rho = 0.5$, they exactly match e_k whereas for $\rho = 1$, they match the coin with the highest e_k

The values of e_k across the whole sampling period are shown in the upper tiles of Fig. 3. An interesting observation is the similarity in e_k between BTC and BCH that use the same mining technology (cf. Theorem 3.10 of [72]).

When ρ is close to zero, the miner maximizes their risk diversification (the miner essentially ignores any other input data) and splits their budget equally among the four coins. However, for larger values of ρ , the miner's decisions are driven by e_k . For $\rho = 0.5$, the miner's allocations precisely coincide with e_k and as ρ increases above 0.5, and ultimately reaches 1, the miner allocates all their resources to the cryptocurrency with the highest e_k .

The allocations in Fig. 3 suggest that risk diversification can have a dramatic impact not only on miner's individual allocations but also on the mining supply of each network as a whole. While lower values of ρ (higher risk diversification) result in stable allocations or allocations that follow less volatile patterns, higher values of ρ (lower risk diversification) lead to violent re-allocations in a day-to-day basis. At the upper extreme, i.e., when $\rho = 1$, a slight change in e_{ik} , i.e., in the efficiency of mining coin k from the perspective of miner i , may lead to a massive re-allocation of their resources. Such fluctuations pose an immediate threat to system stability and naturally raise the question of how such behavior can be curbed in practice.⁴ This brings us to the next set of our empirical analysis which concerns miner's utility maximization and the effects of factor mobility in miner's behavior.

Utility Maximization and Factor Mobility Costs As we saw in Fig. 3, implementing the optimal allocations for high values of ρ (low risk diversification) entails full mobility of resources between different cryptocurrencies in a daily basis. However, such movements typically entail additional switching costs involving transaction costs (e.g., exchanging tokens in staking) or liquidation and new acquisitions of mining equipment. In this part, we model such costs and argue about their effects on miner's utility.

Figure 4 reports metrics on a miner's equilibrium profits for a budget of 1 USD. The first tile shows that the average daily profits (over the whole period of $T = 1358$ days) increase in a sigmoid shape as ρ reaches 1. However, the same holds for the standard deviation of the profits (second tile). As mentioned above, while this strategy (at $\rho = 1$) maximizes the utility of the miner (revenue minus expenses, cf. Eq. (3)), it does so without accounting for restrictions or costs in factor mobility.

To address this issue, we plot in the third tile the *aggregate factor mobility cost* F of miner i ,

$$F := \sum_{t=1}^{T-1} \left[|w_i(t+1) - w_i(t)| + \sum_{k \in M} |b_{i,k}(t+1) - b_{i,k}(t)| \right], \quad (5)$$

i.e., the sum of the differences between allocations over all coins (and unspent budget) between consecutive days during the sample period.⁵ The rightmost (4th) tile shows the aggregate profits over the whole period after accounting for the factor mobility cost, i.e.,

⁴ In particular, fluctuations in mining resources, e.g., hashrate in PoW, cause volatility in cryptocurrency prices [51, 52] which in turn may destabilize blockchain-based applications.

⁵ Different ways to calculate the factor mobility cost led to the same conclusion. Here, we follow one that is standard in portfolio optimization, see e.g., [59].

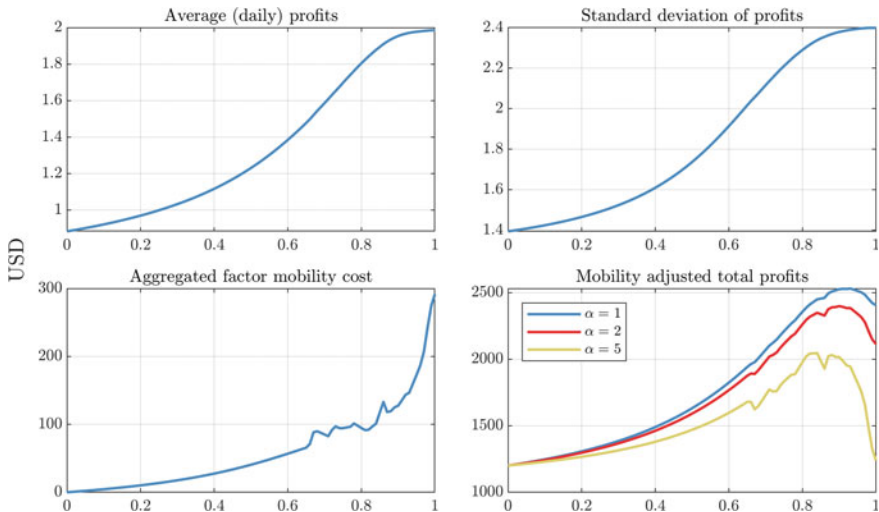


Fig. 4 Miner's utility metrics for all $\rho \in (0, 1]$ (horizontal axis): average and standard deviation of daily profits, factor mobility costs and mobility adjusted total profits for three different per unit switching costs, $\alpha = 1, 2, 5$. Factor mobility costs (cf. Eq. (5)) increase sharply as ρ approaches 1 and mobility adjusted profits (cf. Eq. (6)) are maximized for inner values of ρ

$$u_i^F(\mathbf{b}_i, \mathbf{b}_{-i}) := u_i(\mathbf{b}_i, \mathbf{b}_{-i}) - \alpha F, \quad (6)$$

where $u_i(\mathbf{b}_i, \mathbf{b}_{-i})$ is given by Eq. (3). We plot u_i^F for three different values, $\alpha = 1, 2, 5$, of the switching cost per unit of resource (in USD). The two right-most (3rd and 4th) tiles paint a very different picture about the inefficiencies that materialize as ρ approaches 1 and their effects on utility maximization. The optimal strategy dictates re-allocation of all (for $\rho = 1$) or almost all (for ρ close to 1) of a miner i 's resources to the coin with the highest e_{ik} . Unless this transition is frictionless (in which case, factor mobility costs can be neglected), then the miner's utility is maximized for an *inner* value of $\rho \in (0, 1)$. Along with the previous findings, this suggests that risk diversification is beneficial not only for overall network stability but also for individual miners' utility maximization.

Mining Technologies To further elaborate on the effects of factor mobility costs, we consider a naturally motivated variation of the previous setting. In our data set, BTC and BCH use compatible mining technology and hence, re-allocations between the two coins can be considered costless. Figure 5 shows the factor mobility costs (cf. (5)) after bundling allocations on BTC and BCH. The shape of the mobility costs in the left tile is due to the frequent re-allocations between BTC and BCH after January 2019 and up to July 2020 after which they cease completely (for $\rho = 1$) as ETH starts to consistently dominate the e_k metric (cf. Fig. 3). As was the case with the profits before adjusting for factor mobility costs (first tile of Fig. 4), a miner's utility maximization stands now at odds with network stability. Even for higher values of per unit switching costs ($\alpha = 10$), the miner's profits are maximized at $\rho = 1$ (right tile).

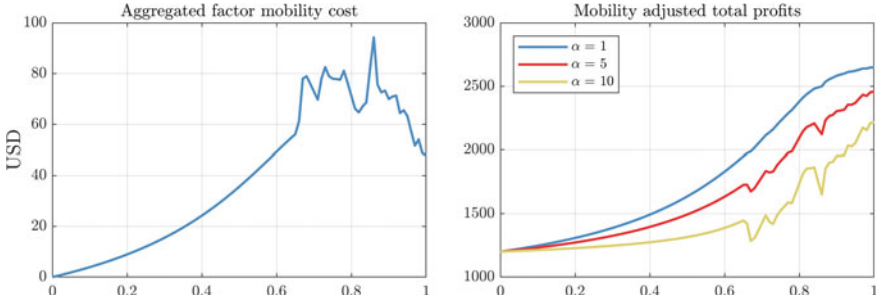


Fig. 5 Same metrics as in the 3rd and 4th tiles of Fig. 4 with higher α 's and no mobility costs between BTC and BCH

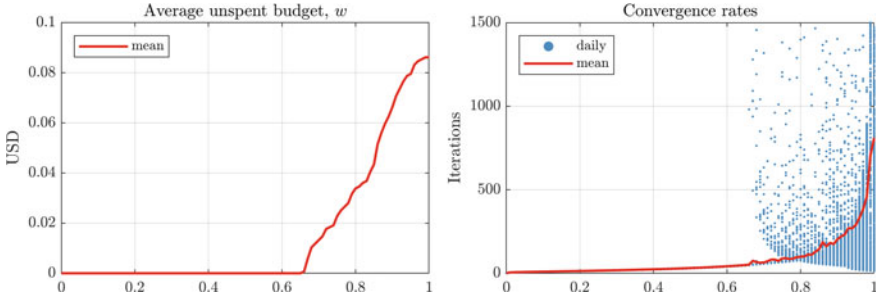


Fig. 6 Average unspent budget (left) and iterations till convergence (right) of the (PR-QCES) protocol in the sample of $T = 1358$ days for all values of $\rho \in (0, 1]$

Unspent Budget The left tile of Fig. 6 reports the average unspent budget (red line) over the sample period for all $\rho \in (0, 1]$. Daily unspent budget (not shown in the graph) is either 0 or very close to 1. For higher risk diversification ($\rho < 0.65$), miners *always* allocate their entire budget. However, miners with higher risk tolerance (ρ closer to 1), tend to mine in less than full capacity more often. This informs the debate on whether miners exhaust their capacities or not [72] and provides new evidence that the answer depends on miners' risk profiles.

Convergence Rates We conclude with metrics concerning the convergence rates of the (PR-QCES) protocol. For each value of $\rho \in (0, 1]$ (horizontal axis), the plot in Fig. 6 shows the iterations (blue dots) and their average (red line) that the algorithm required in each day to converge. The vertical axis is truncated at 1500 iterations (for ρ close to 1 several runs required more than 10k to reach our convergence (stopping) condition which entails five consecutive updates with change less than 10^{-10} ; other stopping rules produced very similar outcomes.) The results, exponentially declining convergence rates for lower values of ρ provide an empirical verification (also in the case of quasi-CES) utilities of existing theoretical results regarding convergence rates of the PR dynamics in the case of Fisher markets with CES utilities [23].

5 Conclusions

The above findings offered a novel *market* perspective on the dynamic nature and growing size of the blockchain mining networks, the increasing degrees of inhomogeneity across the mining population and the effects of risk diversification and factor mobility costs on miners' strategic decisions. The proportional response dynamic produced a naturally motivated, practically relevant and, importantly, globally convergent (to market equilibrium) learning protocol that can be useful both to miners (to optimally allocate their resources in a day-to-day (or less frequent) basis) and to researchers studying miners' incentives. Our results provided evidence that increasing network size and risk diversification among blockchains have beneficial effects in curbing violent resource re-allocations and in aligning miners' incentives for profit maximization with the ecosystem's long-term goals of stability and security.

While useful to miners, blockchain stakeholders and protocol designers, our findings also give rise to interesting questions from a research perspective. Apart from informing technological and managerial decisions, our results provide the starting point and proper framework to address several open questions including the effects of taxation or other kinds of regulation on the blockchain ecosystem [61], the implications of stablecoins in the future of cryptocurrencies [16] and the effects on miners' welfare and network stability of newly designed transaction fee markets (e.g., Ethereum's celebrated EIP-1559 [41, 56, 67]) among others.

A Technical Materials: Proof of Theorem 1

Before we proceed with our empirical results in Sect. 4, we provide the technical details of the proof of Theorem 1. Our proof of Theorem 1 consists of two steps. The first involves the formulation of a *convex program* in the spending (budget) domain that captures the market equilibrium (ME) of the underlying blockchain mining economy with quasi-CES utilities. The second involves the derivation of a general *Mirror Descent* (MD) protocol which converges to the optimal solution of the convex program from the previous step. The proof concludes by showing that the (PR-QCES) protocol is an instantiation of this MD protocol.

Part I: Convex Program Framework. This part utilizes the convex optimization framework in the study of Fisher markets with linear or quasi-linear utilities, see e.g., [6, 23, 25]. As with quasi-linear utilities, the main challenge in this case is to *guess* a convex program that correctly captures the market equilibria also for general quasi-CES utilities [31, 35]. The convex optimization framework that we use to capture the ME spending in quasi-CES Fisher markets is summarized in Fig. 7. Our starting point is a Shmyrev-type convex program proposed by [23] which captures the ME spending in the case of quasi-linear utilities. Our first task is to appropriately modify

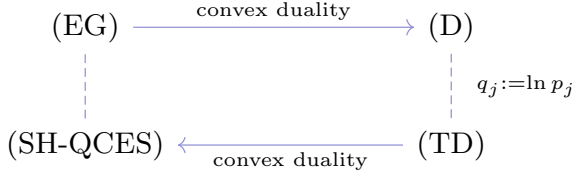


Fig. 7 Convex optimization framework in Theorem 1. The Eisenberg-Gale (EG) [39], its dual (D) and transformed dual (TD) and the Shmyrev-type convex program (SH-QCES) which captures the market equilibrium spending [68]

it so that it captures the ME spending of a quasi-CES Fisher market. The resulting convex program is

$$\begin{aligned} \min F(\mathbf{b}, \mathbf{w}) \text{ s.t. } & \sum_{i=1}^n b_{ik} = p_k, & \forall k \in M, \\ & \sum_{k=1}^m b_{ik} + w_i = K_i, & \forall i \in N, \\ & b_{ik}, w_i \geq 0, & \forall i \in N, k \in M, \end{aligned} \quad (\text{SH-QCES})$$

where $F(\mathbf{b}, \mathbf{w})$ is the following function:

$$\begin{aligned} F(\mathbf{b}, \mathbf{w}) := & - \sum_{i=1}^n \frac{1}{\rho_i} \sum_{k=1}^m b_{ij} \ln[(v_k)^{\rho_i} (b_{ij})^{\rho_i - 1}] + \sum_{k=1}^m p_k \ln p_k \\ & + \sum_{i=1}^n \left[w_i + \frac{\rho_i - 1}{\rho_i} \cdot (K_i - w_i) \ln(K_i - w_i) \right]. \end{aligned}$$

Here, we used the notation $\mathbf{w} = (w_i)_{i \in N}$ and for more clarity, we also wrote $\mathbf{p} = (p_k)_{k \in M}$ to denote the exogenously given aggregate spending vector (cf. $\tilde{\mathbf{b}}$ in the formulation of Theorem 1). While the first and second constraints in (SH-QCES) fully determine the values of p_k and w_i in terms of b_{ij} , it will be more instructive to retain them as separate variables in this part. Our main task is to show that the solutions of (SH-QCES) are solutions to our initial problem, i.e., that they correspond to the ME spending of Γ .

Lemma 1 *The unique minimum point of (SH-QCES) corresponds to the unique market equilibrium spending of Γ .*

Proof We verify that the optimality condition of the convex program (SH-QCES) is the same as the market equilibrium condition. The claim is immediate for $\rho_i = 1$, so we restrict attention to $\rho_i < 1$. To determine the optimality condition of (SH-QCES), we take the partial derivatives of F with respect to b_{ij} and w_i

$$\begin{aligned}\frac{\partial}{\partial b_{ij}} F(\mathbf{b}, \mathbf{w}) &= \frac{1}{\rho_i} (1 - \rho_i \ln v_k) + \frac{1 - \rho_i}{\rho_i} \cdot \ln b_{ij} + \ln p_k \\ \frac{\partial}{\partial w_i} F(\mathbf{b}, \mathbf{w}) &= \frac{1}{\rho_i} [1 - (\rho_i - 1) \ln(K_i - w_i)].\end{aligned}$$

Since $(1 - \rho_i)/\rho_i > 0$, $\lim_{b_{ij} \searrow 0} \frac{1 - \rho_i}{\rho_i} \cdot \ln b_{ij} = -\infty$. Hence, at each minimum point, b_{ij} must be strictly positive. In turn, since b_{ij} is in the relative interior of the domain at each minimum point, and we have the constraint $\sum_{k=1}^m b_{ij} \leq K_i$, it must hold that all $\frac{\partial}{\partial b_{ij}} F(\mathbf{b}, \mathbf{w})$ are identical for all $k \in M$. Equivalently, $\frac{(v_k)^{\rho_i} (b_{ij})^{\rho_i - 1}}{(p_k)^{\rho_i}}$ are identical for all k . Thus, depending on whether $w_i > 0$ or $w_i = 0$, we have the following two cases. If $K_i > w_i > 0$, then $\frac{\partial}{\partial b_{ij}} F(\mathbf{b}, \mathbf{w}) = \frac{\partial}{\partial w_i} F(\mathbf{b}, \mathbf{w})$, which implies $\frac{(v_k)^{\rho_i} (b_{ij})^{\rho_i - 1}}{(p_k)^{\rho_i}} = (K_i - w_i)^{\rho_i - 1}$ for all $k \in M$. If $w_i = 0$, then $\frac{\partial}{\partial b_{ij}} F(\mathbf{b}, \mathbf{w}) \leq \frac{\partial}{\partial w_i} F(\mathbf{b}, \mathbf{w})$, which implies $\frac{(v_k)^{\rho_i} (b_{ij})^{\rho_i - 1}}{(p_k)^{\rho_i}} \geq (K_i - w_i)^{\rho_i - 1}$ for all k .

To determine the ME condition, we need to find the rate of change in a miner's utility w.r.t. changes in spending on mining cryptocurrency k . Due to the cost-homogeneity assumption, in (3), $v_{ij} = v_k/p_k$. Since aggregate expenditures \mathbf{p} are considered as independent signals in the market, the rate is

$$\frac{\partial}{\partial b_{ik}} u_i(\mathbf{b} \mid \mathbf{p}) = \left(\sum_{k=1}^m \frac{(v_k)^{\rho_i} (b_{ik})^{\rho_i}}{(p_k)^{\rho_i}} \right)^{1/\rho_i - 1} \cdot \frac{(v_k)^{\rho_i} (b_{ik})^{\rho_i - 1}}{(p_k)^{\rho_i}} - 1.$$

Since $\rho_i - 1 < 0$ and, hence, $\lim_{b_{ik} \searrow 0} (b_{ik})^{\rho_i - 1} = +\infty$, at the market equilibrium, b_{ik} must be strictly positive. Thus, at the ME, each b_{ij} is in the relative interior of the domain. Together with the constraint $\sum_k b_{ij} \leq K_i$, this implies that $\frac{\partial}{\partial b_{ik}} u_i(\mathbf{b} \mid \mathbf{p})$ are identical for all $k \in M$, which in turn implies that $\frac{(v_k)^{\rho_i} (b_{ij})^{\rho_i - 1}}{(p_k)^{\rho_i}}$ must be identical for all $k \in M$. We denote this (common) value by z_i . Then

$$\begin{aligned}\frac{\partial}{\partial b_{ik}} u_i(\mathbf{b}; \mathbf{p}) &= \left(\sum_{k=1}^m z_i b_{ik} \right)^{1/\rho_i - 1} \cdot z_i - 1 \\ &= (z_i)^{1/\rho_i} \left(\sum_{k=1}^m b_{ij} \right)^{1/\rho_i - 1} - 1 \\ &= (z_i)^{1/\rho_i} (K_i - w_i)^{1/\rho_i - 1} - 1.\end{aligned}$$

Again, there are two cases. If $K_i > w_i > 0$, i.e., if w_i is in the relative interior of its domain, then the above derivative must be zero, i.e., $z_i = (K_i - w_i)^{\rho_i - 1}$ for all i . If $w_i = 0$, then the above derivative at ME is positive or zero, i.e., $z_i \geq (K_i - w_i)^{\rho_i - 1}$ for all i . \square

Part II: From Mirror Descent to Proportional Response. As mentioned above, a useful observation in (SH-QCES) is that the first and second constraints determine the values of p_k, w_i in terms of the b_{ij} 's. Thus, we may rewrite F as a function

of \mathbf{b} only. Then, the convex program (SH-QCES) has only the variables \mathbf{b} , and the only constraints on \mathbf{b} are $b_{ij} \geq 0$ and $\sum_{k=1}^m b_{ij} \leq K_i$. Using this formulation, we can conveniently compute a ME spending by using standard optimization methods like Mirror Descent (MD). Our task in this part, will be to show that the objective function, $F(\mathbf{b})$, of (SH-QCES) is *1-Bregman convex* which implies convergence of the MD protocol and hence, of the (PR-QCES) protocol. To begin, we introduce some minimal additional notation and recap a general result about MD [6, 17] below.

Let C be a compact and convex set. The *Bregman divergence*, d_h , generated by a convex regularizer function h is defined as

$$d_h(\mathbf{b}, \mathbf{a}) := h(\mathbf{b}) - [h(\mathbf{a}) + \langle \nabla h(\mathbf{a}), \mathbf{b} - \mathbf{a} \rangle]. \quad (7)$$

for any $\mathbf{b} \in C$, $\mathbf{a} \in \text{rint}(C)$ where $\text{rint}(C)$ is the relative interior of C . Due to convexity of the function h , $d_h(\mathbf{b}, \mathbf{a})$ is convex in \mathbf{b} , and its value is always non-negative. The *Kullback-Leibler divergence* (KL-divergence) between \mathbf{b} and \mathbf{a} is $\text{KL}(\mathbf{b} \parallel \mathbf{a}) := \sum_k b_k \cdot \ln \frac{b_k}{a_k} - \sum_k b_k + \sum_k a_k$, which is same as the Bregman divergence d_h with regularizer $h(\mathbf{b}) := \sum_k (b_k \cdot \ln b_k - b_k)$. A function f is *L-Bregman convex* w.r.t. Bregman divergence d_h if for any $\mathbf{b} \in C$ and $\mathbf{a} \in \text{rint}(C)$,

$$f(\mathbf{a}) + \langle \nabla f(\mathbf{a}), \mathbf{b} - \mathbf{a} \rangle \leq f(\mathbf{b}) \leq f(\mathbf{a}) + \langle \nabla f(\mathbf{a}), \mathbf{b} - \mathbf{a} \rangle + L \cdot d_h(\mathbf{b}, \mathbf{a}).$$

For the problem of minimizing a convex function $f(\mathbf{b})$ subject to $\mathbf{b} \in C$, the Mirror Descent (MD) method w.r.t. Bregman divergence d_h is given by the update rule in Algorithm 2. In the MD update rule, $1/\xi > 0$ is the step-size, which may vary with t (and typically diminishes with t). However, in the current application of distributed dynamics, time-varying step-size and thus, update rule is *undesirable* or even *impracticable* since this will require from the agents/firms to keep track with a global clock.

Algorithm 2 MD w.r.t. Bregman-divergence d_h

- 1: **procedure** MIRRORDESCENT(f, C, ξ, d_h)
 - 2: **Initialize:** $\mathbf{b}(0) \in C$
 - 3: **while** $t > 0$, $\mathbf{b}(t), \mathbf{b} \in C$ **do**
 - 4: $g(\mathbf{b}, \mathbf{b}(t)) \leftarrow \langle \nabla f(\mathbf{b}(t)), \mathbf{b} - \mathbf{b}(t) \rangle + d_h(\mathbf{b}, \mathbf{b}(t))/\xi$
 - 5: $\mathbf{b}(t+1) \leftarrow \arg \min_{\mathbf{b} \in C} \{g(\mathbf{b}, \mathbf{b}(t))\}$
-

Theorem 2 ([6]) *Suppose that f is an L-Bregman convex function w.r.t. d_h and let $\mathbf{b}(T)$ be the point reached after T applications of the MD update rule in Algorithm 2 with parameter $\xi = 1/L$. Then*

$$f(\mathbf{b}(T)) - f(\mathbf{b}^*) \leq L \cdot d_h(\mathbf{b}^*, \mathbf{b}(0))/T.$$

Using the above, we are now ready to show that the objective function of the (SH-QCES) is a 1-Bregman convex function w.r.t. the KL-divergence. This is the statement of Lemma 2. Its proof closely mirrors an analogous statement in [23] and is, thus, omitted.

Lemma 2 *The objective function F of (SH-QCES) is a 1-Bregman convex function w.r.t. the divergence $\sum_{i=1}^n \frac{1}{\rho_i} \cdot \text{KL}(x'_i || x_i)$.*

We now turn to the derivation of the (PR-QCES) protocol from the MD algorithm for a suitable choice of ξ . For the convex program (SH-QCES), the MD rule (Algorithm 2) is

$$\begin{aligned} (\mathbf{b}(t+1), \mathbf{w}(t+1)) &= \arg \min_{(\mathbf{b}, \mathbf{w}) \in C} \left\{ \sum_{i=1}^n \frac{1}{\rho_i} \cdot \text{KL}(\mathbf{b}_i || \mathbf{b}_i(t)) + \right. \\ &+ \sum_{i=1}^n \sum_{k=1}^m \frac{(b_{ij} - b_{ij}(t))}{\rho_i} \cdot \left(1 - \ln \frac{(v_k)^{\rho_i} b_{ij}(t)^{\rho_i-1}}{p_k(t)^{\rho_i}} \right) \\ &\left. + \sum_{i=1}^n \frac{1}{\rho_i} [1 - (\rho_i - 1) \cdot \ln(K_i - w_i(t))] \cdot (w_i - w_i(t)) \right\}. \end{aligned}$$

Since $\sum_{k=1}^m b_{ij} + w_i$ is constant in the domain C , we may ignore any term that does not depend on \mathbf{b} and \mathbf{w} , and any positive constant in the objective function and simplify the above update rule to

$$\begin{aligned} (\mathbf{b}(t+1), \mathbf{w}(t+1)) &= \arg \min_{(\mathbf{b}, \mathbf{w}) \in C} \left\{ \sum_{i=1}^n (1 - \rho_i) \ln(K_i - w_i(t)) \cdot w_i \right. \\ &- \left. \sum_{i=1}^n \sum_{k=1}^m \left(\ln \frac{(v_k)^{\rho_i} b_{ij}(t)^{\rho_i-1}}{p_k(t)^{\rho_i}} \cdot b_{ij} - b_{ij} \ln \frac{b_{ij}}{b_{ij}(t)} + b_{ij} \right) \right\} \\ &\triangleq \arg \min_{(\mathbf{b}, \mathbf{w}) \in C} \bar{F}(\mathbf{b}, \mathbf{w}). \end{aligned}$$

Concerning the partial derivatives of \bar{F} , we have

$$\begin{aligned} \frac{\partial}{\partial b_{ij}} \bar{F}(\mathbf{b}, \mathbf{w}) &= \ln b_{ij} - \ln \frac{(v_k)^{\rho_i} b_{ij}(t)^{\rho_i}}{p_k(t)^{\rho_i}} \\ \frac{\partial}{\partial w_i} \bar{F}(\mathbf{b}, \mathbf{w}) &= (1 - \rho_i) \ln(K_i - w_i(t)). \end{aligned}$$

As before, for each fixed i , the values of $\ln b_{ij} - \ln \frac{(v_k)^{\rho_i} b_{ij}(t)^{\rho_i}}{p_k(t)^{\rho_i}}$ for all $k \in M$ are identical. In other words, there exists $c_i > 0$ such that $b_{ij} = c_i \cdot (v_k)^{\rho_i} b_{ij}(t)^{\rho_i} / p_k(t)^{\rho_i}$. As before, there are two cases which depend on

$$S_i \triangleq \sum_{k \in M} (v_k)^{\rho_i} b_{ij}(t)^{\rho_i} / p_k(t)^{\rho_i} = \sum_{k \in M} (v_{ik} b_{ik})^{\rho_i}.$$

If $S_i \geq K_i \cdot (K_i - w_i(t))^{\rho_i - 1}$, then we set $b_{ij}(t + 1) = K_i \cdot (v_{ik}b_{ik})/S_i$, for each $k \in M$, and $w_i(t + 1) = 0$. At this point, we have

$$\frac{\partial}{\partial b_{ij}} \bar{F}(\mathbf{b}, \mathbf{w}) = \ln \frac{K_i}{S_i} \leq \frac{\partial}{\partial w_i} \bar{F}(\mathbf{b}, \mathbf{w}),$$

so the optimality condition is satisfied. If $S_i < K_i \cdot (K_i - w_i(t))^{\rho_i - 1}$, then, we set $b_{ij}(t + 1) = (K_i - w_i(t))^{1 - \rho_i} \cdot (v_{ik}b_{ik})^{\rho_i}$, for each k , and $w_i(t + 1) = K_i - \sum_{k=1}^m b_{ij}(t + 1) > 0$. At this point, we have $\frac{\partial}{\partial b_{ij}} \bar{F}(\mathbf{b}, \mathbf{w}) = \frac{\partial}{\partial w_i} \bar{F}(\mathbf{b}, \mathbf{w})$, so the optimality condition is satisfied.

Theorem 2 now guarantees that the updates of (PR-QCES) converge to an optimal solution of (SH-QCES). This shows that the (PR-QCES) dynamics converge to the ME of a Fisher market with quasi-CES utilities for any $0 < \rho_i \leq 1$ and concludes the proof of Theorem 1. Note that the previous proof cannot be extended in a straightforward way to values of $\rho_i < 0$, since in that case, a direct calculation shows that F is neither convex nor concave function which implies that the above argument does not apply.

References

1. Alkalay-Houlihan, C., & Shah, N. (2019). The pure price of anarchy of pool block withholding attacks in bitcoin mining. *AAAI Conference on Artificial Intelligence, AAAI-19* 33(1), 1724–1731. <https://doi.org/10.1609/aaai.v33i01.33011724>.
2. Arnosti, N., & Weinberg, S. M. (2018). Bitcoin: A natural oligopoly. In A. Blum (ed.), *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*. Leibniz International Proceedings in Informatics (LIPIcs) (vol. 124, pp. 5:1–5:1). Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl. <https://doi.org/10.4230/LIPIcs.ITCS.2019.5>.
3. Auer, R. (2019). Beyond the doomsday economics of proof-of-work in cryptocurrencies. Discussion Paper DP13506. London: Centre for Economic Policy Research. https://cepr.org/active/publications/discussion_papers/dp.php?dpno=13506
4. Bailey, J. P., & Piliouras, G. (2018). Multiplicative weights update in zero-sum games. In *Proceedings of the 2018 ACM Conference on Economics and Computation* (p. 321–338) New York, NY, USA: EC '18, Association for Computing Machinery. <https://doi.org/10.1145/3219166.3219235>.
5. Bentov, I., Gabizon, A., & Mizrahi, A. (2016). Cryptocurrencies without proof of work. In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, & K. Rohloff (Eds.), *Financial Cryptography and Data Security* (pp. 142–157). Berlin Heidelberg, Berlin, Heidelberg: Springer.
6. Birnbaum, B., Devanur, N.R., & Xiao, L. (2011). Distributed algorithms via gradient descent for fisher markets. In *Proceedings of the 12th ACM Conference on Electronic Commerce* (p. 127–136). New York, NY, USA: EC '11, Association for Computing Machinery. <https://doi.org/10.1145/1993574.1993594>.
7. Bonneau, J. (2016). Why buy when you can rent? In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, & K. Rohloff (Eds.), *Financial Cryptography and Data Security* (pp. 19–26). Berlin Heidelberg, Berlin, Heidelberg: Springer.
8. Bonneau, J., Miller, A., Clark, J., Narayanan, A., Kroll, J. A., & Felten, E. W. (2015) Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. In *Proceedings of the*

- 2015 IEEE Symposium on Security and Privacy (p. 104–121). USA: SP '15, IEEE Computer Society. <https://doi.org/10.1109/SP.2015.14>
9. Brânzei, S., Devanur, N., & Rabani, Y. (2021). Proportional dynamics in exchange economies. In *Proceedings of the 22nd ACM Conference on Economics and Computation* (p. 180–201). EC '21, Association for Computing Machinery, New York, NY, USA (2021). <https://doi.org/10.1145/3465456.3467644>.
 10. Branzei, S., Mehta, R., & Nisan, N. (2018). Universal growth in production economies. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, R. Garnett (eds.), *Advances in Neural Information Processing Systems* vol. 31. New York, USA: Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2018/file/692f93be8c7a41525c0baf2076aecfb4-Paper.pdf>
 11. Brown-Cohen, J., Narayanan, A., Psomas, A., & Weinberg, S.M. (2019). Formal barriers to longest-chain proof-of-stake protocols. In *Proceedings of the 2019 ACM Conference on Economics and Computation* (p. 459–473). New York, NY, USA: EC'19, ACM. <https://doi.org/10.1145/3328526.3329567>.
 12. Budish, E. (2018). The economic limits of bitcoin and the blockchain. Working Paper 24717, *National Bureau of Economic Research*. <https://doi.org/10.3386/w24717>.
 13. Buterin, V. (2013). A next-generation smart contract and decentralized application platform. <https://ethereum.org/en/whitepaper/>.
 14. Buterin, V., Reijersbergen, D., Leonardos, S., & Piliouras, G. (2019). Incentives in ethereum's hybrid casper protocol. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 236–244). USA: IEEE. <https://doi.org/10.1109/BLOC.2019.8751241>.
 15. Carlsten, M., Kalodner, H., Weinberg, S. M., & Narayanan, A. (2016). On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (p. 154–167). Vienna, Austria: CCS '16, ACM. <https://doi.org/10.1145/2976749.2978408>.
 16. Catalini, C., & Massari, J. (2020). Stablecoins and the future of money. *Harvard Business Review* (online). <https://hbr.org/2021/08/stablecoins-and-the-future-of-money>.
 17. Chen, G., & Teboulle, M. (1993). Convergence analysis of a proximal-like minimization algorithm using bregman functions. *SIAM J. Optim.*, 3(3), 538–543. <https://doi.org/10.1137/0803026>
 18. Chen, J., & Micali, S. (2019). Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777, 155–183. <https://doi.org/10.1016/j.tcs.2019.02.001>
 19. Chen, L., Xu, L., Gao, Z., Sunny, A.I., Kasichainula, K., & Shi, W. (2021). A game theoretical analysis of non-linear blockchain system. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems* (p. 323–331). Richland, SC: AAMAS '21, International Foundation for Autonomous Agents and Multiagent Systems.
 20. Chen, X., Papadimitriou, C., & Roughgarden, T. (2019). An axiomatic approach to block rewards. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (pp. 124–131). New York, NY, USA: AFT '19, ACM. <https://doi.org/10.1145/3318041.3355470>.
 21. Cheung, Y. K., Cole, R., & Devanur, N. R. (2020). Tatonnement beyond gross substitutes? Gradient descent to the rescue. *Games and Economic Behavior*, 123, 295–326. <https://doi.org/10.1016/j.geb.2019.03.014>
 22. Cheung, Y.K., Cole, R., & Rastogi, A. (2012). Tatonnement in ongoing markets of complementary goods. In *Proceedings of the 13th ACM Conference on Electronic Commerce* (pp. 337–354). New York, NY, USA: EC '12, Association for Computing Machinery. <https://doi.org/10.1145/2229012.2229039>.
 23. Cheung, Y.K., Cole, R., & Tao, Y. (2018). Dynamics of distributed updating in fisher markets. In (pp. 351–368). New York, NY, USA: EC'18, ACM. <https://doi.org/10.1145/3219166.3219189>.
 24. Cheung, Y.K., Hoefer, M., & Nakhe, P. (2019). Tracing equilibrium in dynamic markets via distributed adaptation. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 1225–1233). Richland, SC: AAMAS '19, International Foundation for Autonomous Agents and Multiagent Systems.

25. Cheung, Y.K., Leonardos, S., & Piliouras, G. (2021). Learning in markets: Greed leads to chaos but following the price is right. In Z. H. Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21* (pp. 111–117). International Joint Conferences on Artificial Intelligence Organization, virtual. <https://doi.org/10.24963/ijcai.2021/16>.
26. Cheung, Y.K., & Piliouras, G. (2019). Vortices instead of equilibria in minmax optimization: Chaos and butterfly effects of online learning in zero-sum games. In *Conference on Learning Theory, COLT 2019* (pp. 25–28, 807–834). Phoenix, AZ, USA. <http://proceedings.mlr.press/v99/cheung19a.html>
27. Cheung, Y.K., & Piliouras, G. (2020). Chaos, extremism and optimism: Volume analysis of learning in games. *CoRR* **abs/2005.13996**. <https://arxiv.org/abs/2005.13996>
28. Cheung, Y.K., & Tao, Y. (2021). Chaos of learning beyond zero-sum and coordination via game Decompositions. In ICLR.
29. Cheung, Y.K. (2018). Multiplicative weights updates with constant step-size in graphical constant-sum games. In (pp. 3532–3542). NeurIPS.
30. Chiu, J., & Koepl, T.V. (2018). Incentive compatibility on the blockchain. Bank of Canada Staff Working Paper 2018–34. Ottawa: Bank of Canada.
31. Cole, R., Devanur, N., Gkatzelis, V., Jain, K., Mai, T., Vazirani, V. V., & Yazdanbod, S. (2017). Convex program duality, fisher markets, and nash social welfare. In *Proceedings of the 2017 ACM Conference on Economics and Computation* (pp. 459–460). New York, NY, USA: EC '17, Association for Computing Machinery. <https://doi.org/10.1145/3033274.3085109>.
32. Cole, R., & Tao, Y. (2016). Large market games with near optimal efficiency. In (pp. 791–808). New York, NY, USA: EC'16. ACM. <https://doi.org/10.1145/2940716.2940720>.
33. De Vries, A. (2018). *Bitcoin's growing energy problem*. *Joule*, 2(5), 801–805. <https://doi.org/10.1016/j.joule.2018.04.016>
34. De Vries, A. (2020). Bitcoin's energy consumption is underestimated: A market dynamics approach. *Energy Research & Social Science*, 70, 101721. <https://doi.org/10.1016/j.erss.2020.101721>
35. Devanur, N. R. (2009). Fisher markets and convex programs. Published online. <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.177.351&rep=rep1&type=pdf>.
36. Dimitri, N. (2017). *Bitcoin mining as a contest*. *Ledger*, 2, 31–37. <https://doi.org/10.5195/ledger.2017.96>
37. Dvijotham, K., Rabani, Y., & Schulman, L. J. (2020). Convergence of incentive-driven dynamics in Fisher markets. *Games and Economic Behavior in press, corrected proof, online*. <https://doi.org/10.1016/j.geb.2020.11.005>
38. Easley, D., O'Hara, M., & Basu, S. (2019). From mining to markets: The evolution of bitcoin transaction fees. *Journal of Financial Economics*, 134(1), 91–109. <https://doi.org/10.1016/j.jfineco.2019.03.004>
39. Eisenberg, E., & Gale, D. (1959). Consensus of subjective probabilities: The pari-mutuel method. *Ann. Math. Statist.*, 30(1), 165–168.
40. Eyal, I., & Sirer, E. G. (2018). Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7), 95–102. <https://doi.org/10.1145/3212998>
41. Ferreira, M. V. X., Moroz, D. J., Parkes, D. C., & Stern, M. (2021). Dynamic posted-price mechanisms for the blockchain transaction-fee market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies* (pp. 86–99). New York, NY, USA: AFT '21, Association for Computing Machinery. <https://doi.org/10.1145/3479722.3480991>.
42. Fiat, A., Karlin, A., Koutsoupias, E., & Papadimitriou, C. (2019). Energy equilibria in proof-of-work mining. In (pp. 489–502). New York, NY, USA: EC'19, ACM. <https://doi.org/10.1145/3328526.3329630>.
43. Gandal, N., & Gans, J. (2019). More (or less) economic limits of the blockchain. Discussion Paper DP14154, London, Centre for Economic Policy Research. https://cepr.org/active/publications/discussion_papers/dp.php?dpno=14154.

44. Gao, Y., & Kroer, C. (2020). First-order methods for large-scale market equilibrium computation. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, & H. Lin (Eds.), *Advances in Neural Information Processing Systems* (vol. 33, pp. 21738–21750). Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2020/file/f75526659f31040afeb61cb7133e4e6d-Paper.pdf>.
45. Garay, J., Kiayias, A., & Leonardos, N. (2015). The bitcoin backbone protocol: Analysis and applications. In E. Oswald, & M. Fischlin (Eds.), *Advances in Cryptology—EUROCRYPT* (pp. 281–310). Berlin: Springer. https://doi.org/10.1007/978-3-662-46803-6_10.
46. Gersbach, H., Mamageishvili, A., & Schneider, M. (2020). Vote delegation and malicious parties. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 1–2). <https://doi.org/10.1109/ICBC48266.2020.9169391>.
47. Gersbach, H., Mamageishvili, A., & Schneider, M. (2022). Staking pools on blockchains. <https://doi.org/10.48550/ARXIV.2203.05838>.
48. Goren, G., & Spiegelman, A. (2019). Mind the mining. In *Proceedings of the 2019 ACM Conference on Economics and Computation* (pp. 475–487). New York, NY: EC '19, ACM. <https://doi.org/10.1145/3328526.3329566>.
49. Huberman, G., Leshno, J. D., & Moallemi, C. (2021). Monopoly without a monopolist: An economic analysis of the Bitcoin payment system. *The Review of Economic Studies*, 1–30., <https://doi.org/10.1093/restud/rdab014.rdab014>
50. Kiayias, A., Koutsoupias, E., Kyropoulou, M., & Tselekounis, Y. (2016). Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation* (pp. 365–382). New York, NY: EC '16, ACM. <https://doi.org/10.1145/2940716.2940773>.
51. Koki, C., Leonardos, S., & Piliouras, G. (2020). Do cryptocurrency prices camouflage latent economic effects? A bayesian hidden markov approach. *Future Internet*, 28(1), 5. <https://doi.org/10.3390/fi12030059>
52. Koki, C., Leonardos, S., & Piliouras, G. (2022). Exploring the predictability of cryptocurrencies via Bayesian hidden Markov models. *Research in International Business and Finance*, 59, 101554. <https://doi.org/10.1016/j.ribaf.2021.101554>
53. Kwon, Y., Liu, J., Kim, M., Song, D., & Kim, Y. (2019). Impossibility of full decentralization in permissionless blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (pp. 110–123). New York, NY: AFT '19, Association for Computing Machinery. <https://doi.org/10.1145/3318041.3355463>.
54. Leonardos, N., Leonardos, S., & Piliouras, G. (2020). Oceanic games: Centralization risks and incentives in blockchain mining. In P. Pardalos, I. Kotsireas, Y. Guo, & W. Knottenbelt (Eds.), *Mathematical Research for Blockchain Economy* (pp. 183–199). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-37110-4_13.
55. Leonardos, S., & Melolidakis, C. (2020). Endogenizing the cost parameter in cournot oligopoly. *International Game Theory Review*, 22(02), 2040004. <https://doi.org/10.1142/S0219198920400046>
56. Leonardos, S., Monnot, B., Reijsbergen, D., Skoulakis, S., & Piliouras, G. (2021). Dynamical analysis of the EIP-1559 Ethereum Fee Market. In *Proceedings of the 3rd ACM conference on Advances in Financial Technologies* (p. online). New York, NY: AFT '21, Association for Computing Machinery.
57. Leonardos, S., Reijsbergen, D., & Piliouras, G. (2019). Weighted voting on the blockchain: Improving consensus in proof of stake protocols. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 376–384). <https://doi.org/10.1109/BLOC.2019.8751290>.
58. Leonardos, S., Reijsbergen, D., & Piliouras, G. (2020). PREStO: A systematic framework for blockchain consensus protocols. *IEEE Transactions on Engineering Management*, 67(4), 1028–1044. <https://doi.org/10.1109/TEM.2020.2981286>
59. Liu, W. (2019). Portfolio diversification across cryptocurrencies. *Finance Research Letters*, 29, 200–205. <https://doi.org/10.1016/j.frl.2018.07.010>
60. M. Shen. (2020). Crypto investors have ignored three straight 51% attacks on ETC. Accessed February 11, 2021, from www.coindesk.com.

61. Marple, T. (2021). Bigger than Bitcoin: A theoretical typology and research agenda for digital currencies. *Business and Politics, online*, 1–17,. <https://doi.org/10.1017/bap.2021.12>
62. Monnot, B., Hum, Q., Koh, C. M., & Piliouras, G. (2020). Ethereum's transaction fee market reform in eip 1559. WINE 20, Workshop on Game Theory in Blockchain. www.workshop.com.
63. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. published online. <https://bitcoin.org/bitcoin.pdf>
64. Noda, S., Okumura, K., & Hashimoto, Y. (2020). An economic analysis of difficulty adjustment algorithms in proof-of-work blockchain systems. In *Proceedings of the 21st ACM Conference on Economics and Computation* (p. 611). New York, NY: EC '20, Association for Computing Machinery. <https://doi.org/10.1145/3391403.3399475>.
65. Palaiopanos, G., Panageas, I., & Piliouras, G. (2017). Multiplicative weights update with constant step-size in congestion games: Convergence, limit cycles and chaos. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems* vol. 30. Curran Associates, Inc. <https://proceedings.neurips.cc/paper/2017/file/e93028bdc1aacdfb3687181f2031765d-Paper.pdf>.
66. Reijsbergen, D., Sridhar, S., Monnot, B., Leonardos, S., Skoulakis, S., & Piliouras, G. (2021). Transaction fees on a honeymoon: Ethereum's eip-1559 one month later. In *2021 IEEE International Conference on Blockchain (Blockchain)* (pp. 196–204). <https://doi.org/10.1109/Blockchain53845.2021.00034>.
67. Roughgarden, T. (2021). *Transaction fee mechanism design. SIGecom Exch.*, 19(1), 52–55. <https://doi.org/10.1145/3476436.3476445>
68. Shmyrev, V. I. (2009). An algorithm for finding equilibrium in the linear exchange model with fixed budgets. *Journal of Applied and Industrial Mathematics*, 3(4), 505. <https://doi.org/10.1134/S1990478909040097>
69. Singh, R., Dwivedi, A. D., Srivastava, G., Wiszniewska-Matyszek, A., & Cheng, X. (2020). A game theoretic analysis of resource mining in blockchain. *Cluster Computing*, 23(3), 2035–2046. <https://doi.org/10.1007/s10586-020-03046-w>
70. State of the Dapps. (2021). Explore decentralized applications. Accessed February 11, 2021, from www.stateofthedapps.com.
71. Stoll, C., Klafen, L., & Gellersdörfer, U. (2019). *The carbon footprint of bitcoin. Joule*, 3(7), 1647–1661. <https://doi.org/10.1016/j.joule.2019.05.012>
72. Sun, J., Tang, P., & Zeng, Y. (2020). Games of miners. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 1323–1331). AAMAS '20, International Foundation for Autonomous Agents and Multiagent Systems, Richland, SC. <https://doi.org/10.5555/3398761.3398914>.
73. Wave Financial LLC. (2021). Ethereum 2.0 staking, a worthwhile investment?. Accessed February 11, 2021, from www.cityam.com.
74. Werner, S. M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., & Knottenbelt, W. J. (2021). SoK: Decentralized Finance (DeFi).
75. Wu, F., & Zhang, L. (2007). Proportional response dynamics leads to market equilibrium. In *Proceedings of the Thirty-ninth Annual ACM Symposium on Theory of Computing* (pp. 354–363). New York, NY: STOC '07, ACM. <https://doi.org/10.1145/1250790.1250844>.
76. Xiao, Y., Zhang, N., Lou, W., & Hou, Y. T. (2020). A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys Tutorials*, 22(2), 1432–1465. <https://doi.org/10.1109/COMST.2020.2969706>
77. Zhang, L. (2009). Proportional response dynamics in the Fisher market. *Theoretical Computer Science*, 412(24), 2691–2698. <https://doi.org/10.1016/j.tcs.2010.06.021>, Selected Papers from 36th International Colloquium on Automata, Languages and Programming (ICALP 2009).

On the Impact of Vote Delegation



Hans Gersbach, Akaki Mamageishvili, and Manvir Schneider

Abstract We examine vote delegation on blockchains where preferences of agents are private information. One group of agents (delegators) does not want to participate in voting and either abstains under conventional voting or can delegate its votes to a second group (voters) who decides between two alternatives. We show that free delegation favors minorities, that is, alternatives that have a lower chance of winning ex-ante. The same occurs if the number of voting rights that actual voters can exert is capped. When the number of delegators increases, the probability that the ex-ante minority wins under free and capped delegation converges to the one under conventional voting—albeit non-monotonically.

Keywords Voting · Delegation · Abstention · Democracy · Blockchain · Governance

This research was partially supported by the Zurich Information Security and Privacy Center (ZISC). We thank Roger Wattenhofer, Huseyin Yildirim, Marcus Pivato, Florian Brandl and participants of the Astute Modeling Seminar at ETH Zurich for their valuable feedback.

H. Gersbach · A. Mamageishvili · M. Schneider (✉)
CER-ETH, ETH Zürich, Zürichbergstrasse 18, 8092 Zürich, Switzerland
e-mail: manvirschneider@ethz.ch

H. Gersbach
e-mail: hgersbach@ethz.ch

A. Mamageishvili
e-mail: amamageishvili@ethz.ch

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes
in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_3

1 Introduction

Many blockchains have already implemented staking or will implement it in the near future. Examples include Cardano,¹ Solana [15], Polkadot [14] and Tezos [7]. These blockchains allow agents to delegate their stakes to other agents for validation purposes or to govern the blockchain. Typically, delegators do not know the preferences of agents to whom they delegate, as every participant is merely embodied by an address in the form of a number or a pseudonym.

In this paper, we examine how voting outcomes are affected when vote delegation is allowed, but delegators do not know the preferences of the proxy to whom they delegate. In particular, we study the following problem. A polity—say a jurisdiction or a blockchain community—decides between two alternatives A and B . A fraction of individuals does not want to participate in the voting. Henceforth, these individuals are called “delegators”. They would abstain in traditional voting and delegate their vote in liquid democracy. There are many reasons why individuals do not want to participate in voting. For example, they might want to avoid the costs of informing themselves or do not want to incur the costs of voting.

Our central assumption is that delegators do not know the preferences of the individuals who will vote, that is, whether these individuals favor A or B . This is obvious in the context of blockchains in which delegators do not know whether voters are honest (interested in the validation of transactions) or dishonest (interested in creating invalid transactions or in disrupting the system as a whole). For standard democracies, our assumption represents a polar case.

We call individuals willing to vote “voters”, individuals favoring A (B) “ A -voters” (“ B -voters”), and we call the group of the A -voters (B -voters) “ A -party” (“ B -party”). From the perspective of delegators, of voters themselves, and of any outsider who might want to design vote delegation, the probability that a voter favors A is some number p ($0 < p < 1$). If $p < \frac{1}{2}$, the chances of alternative A to win under standard voting is positive but below $\frac{1}{2}$. We call the A -party the “ex-ante minority” in such cases.

We compare three voting schemes. First, under standard voting, delegation is not allowed and thus delegators abstain. Second, with free vote delegation, arbitrary delegation is allowed. Since delegators do not know the preferences of voters and all voters are thus alike for them, every voter has the same chance to obtain a vote from delegators. We thus model free delegation as a random process in which delegated voting rights are randomly and uniformly assigned to voters. This can be achieved in one step where delegators know the pool of voters, or by transitive delegation, in which each individual can delegate his/her voting right to some other individual, who, in turn, can delegate the accumulated voting rights to a next individual. This process stops when an individual decides to use the received votes and exert all voting rights received. Third, we introduce capped vote delegation. With capped vote delegation, the number of votes an individual can receive is limited. Essentially, if a voter has

¹ <https://www.cardano.org> (retrieved January 20, 2022). See also [9].

reached the cap of voting rights through delegation, s/he cannot receive any more delegated votes. Thus, only voters who have not reached the cap yet will be able to receive further voting rights.

In our analysis, we model the total number of voters by a random variable with distribution F , whereas F can be any discrete distribution.

We establish three main results: First, compared to standard voting, free delegation favors the minority. This result, which requires an elaborate proof, shows that vote delegation cannot only reverse voting outcomes but also increases the probability that the minority wins.

An essential assumption for this result is that the types of voters (and thus their preferences) are stochastically independent. By a counterexample, we show that delegation can favor the majority if this assumption is *not* made. However, delegation favors the minority in the class of distributions where agents receive stochastically independent signals of their type, such as, for instance, for a Poisson distribution.

Second, the same occurs with a capped delegation. That is, the capped delegation also increases the minority's probability to win. Numerical examples show that the increase is smaller than under free delegation. Third, when the fraction of delegators increases compared to voters, the probability that minorities win under free or capped delegation converges to the same probability as under standard voting. However, we see in the case of a Poisson distribution that the convergence is not monotonic. The results show that outcomes *with delegation* may significantly differ from *standard* election outcomes and probabilistically violate the majority principle, i.e. a majority of citizens who is willing to cast a vote for a particular alternative may lose. Hence, implementing vote delegation might raise the risk that dishonest agents become a majority—which is a major concern for blockchains.

This paper is organized as follows. In Sect. 2, we discuss the related literature. Section 3 introduces our model. In Sect. 4, we show our results for free and capped delegation and study their asymptotic behavior. Section 5 concludes.²

2 Related Literature on Vote Delegation

Some recent papers examine how vote delegation impacts outcomes. Important studies analyzing vote delegation from an algorithmic and AI perspective have developed several delegation rules that allow to examine how many votes a representative can and should obtain for voting in collective decisions³ and whether the delegation of votes to neighbours in a network may deliver more desirable outcomes than conventional voting.⁴ Kahng et al. [8] show that even with a delegation from a less informed

² All proofs can be provided upon request.

³ See Gölz et al. [6] and Kotsialou and Riley [10].

⁴ See Kahng et al. [8]. Gersbach et al. [5] show that delegation delivers a positive outcome with higher probability than conventional voting in costly voting environments and in the presence of malicious voters.

to a better informed voter, the probability of implementing the right alternative is decreasing in a generic network with information acquisition. Our model and results are complementary to Kahng et al. [8]. We show that when preferences are private information, delegation favors minorities. Caragiannis and Micha [3] extend the work of Kahng et al. [8] and show that liquid democracy can yield less desirable outcomes than conventional voting or a dictatorship.

Delegation games on networks were studied by Bloembergen et al. [2] and Escoffier et al. [4]. Bloembergen et al. [2] identify conditions for the existence of Nash equilibria of such games, while Escoffier et al. [4] show that in more general setups, no Nash equilibrium may exist, and it may even be NP -complete to decide *whether* one exists at all. We adopt a collective decision framework and study how free delegation and capped delegation impact outcomes when delegators do not know the preferences of the individuals to whom they delegate.

There is recent work on representative democracy, e.g. Pivato and Soh [12] and Abramowitz and Mattei [1]. The first paper studies a model in which legislators are chosen by voters, and votes are counted according to weight, which is the number of votes a representative possesses. Pivato and Soh [12] show that the voting outcome is the same in a large election as for conventional voting. Soh [13] extends this model by using other forms of voting, such as *Weighted Approval Voting*. Furthermore, there is research on *Flexible Representative Democracy*, see e.g. Abramowitz and Mattei [1]. In their model, experts are elected first and then voters either allocate their vote among the representatives or vote themselves. In our model, delegators delegate their votes uniformly at random to *any* voter in the society. In Flexible Representative Democracy, random uniform delegation can be achieved by uniformly allocating votes among the representatives.

3 Model

We consider a large polity (a society or a blockchain community) that faces a binary choice with alternatives A and B . There is a group of $m \in \mathbb{N}_+$ individuals (called “delegators”) who do not want to vote and either abstain under conventional voting or delegate their voting rights under vote delegation. The remaining population (“voters”) votes. Voters have private information about their preference for A or B . Hence, when a delegator delegates his/her voting right to a voter, it is equivalent to uniformly and randomly delegating a voting right to *one* voter. A voter prefers alternative A (B) with probability p ($1 - p$), where $0 < p < 1$. Without loss of generality, we assume $p > \frac{1}{2}$. Voters favoring A (B) are called “ A -voters” (“ B -voters”) and the respective group is called “ A -party” (“ B -party”).

We assume that the number of voters is given by a random variable with distribution F , that is, the probability that there are i voters is equal to $F(i)$. We compare three voting processes:

- Conventional voting: Each voter casts one vote.

- Free delegation: m delegators randomly delegate their voting rights. All voters exert all voting rights they have.
- Capped delegation: m delegators delegate their voting rights randomly to voters. If a voter has reached the cap, all further delegated voting rights are distributed among the remaining voters, up to each voter's cap. If all voters have reached their cap, superfluous voting rights are destroyed.

We start with conventional voting and denote by $P(p)$ the probability that A wins. It is calculated by the following formula:

$$P(p) = \sum_{i=0}^{\infty} F(i) \sum_{k=0}^i \binom{i}{k} p^k (1-p)^{i-k} g(k, i-k), \quad (1)$$

where $g(k, l)$ is the probability that A -voters are in the majority if there are k A -voters and l B -voters. We calculate $g(k, l)$ as follows:

$$g(k, l) := \begin{cases} 1, & \text{if } k > l, \\ \frac{1}{2}, & \text{if } k = l, \\ 0, & \text{if } k < l. \end{cases}$$

The first sum in Eq. (1) corresponds to sampling the number of voters i . The second sum corresponds to sampling the number of A -voters k out of total i voters.

In the case of free delegation, the probability that A wins, denoted by $P(p, m)$, is equal to:

$$P(p, m) = \sum_{i=0}^{\infty} F(i) \sum_{k=0}^i \binom{i}{k} p^k (1-p)^{i-k} G(k, i-k, m), \quad (2)$$

where $G(k, l, m)$ denotes the probability that A -voters win if there are k A -voters, l B -voters and m voters delegate their votes. The value is calculated in the following way:

$$G(k, l, m) = \begin{cases} \frac{1}{2} & \text{if } k = l = 0, \\ \sum_{h=0}^m \binom{m}{h} \left(\frac{k}{k+l}\right)^h \left(\frac{l}{k+l}\right)^{m-h} g(k+h, l+m-h) & \text{else.} \end{cases}$$

For $k > 0$ or $l > 0$, $G(k, l, m)$ is calculated by sampling the number of votes h out of m votes that are delegated to k A -voters and $m - h$ votes are delegated to l B -voters.

We now consider the capped delegation procedure, where delegated votes are distributed randomly among the other voters, with the restriction that no voter can have more than c votes. If a voter already has c votes, s/he is not allowed to receive more. With cap c , the probability that A -voters have a majority, denoted as $P_c(p, m)$, is equal to

$$P_c(p, m) = \sum_{i=0}^{\infty} F(i) \sum_{k=0}^i \binom{i}{k} p^k (1-p)^{i-k} G_c(k, i-k, m), \quad (3)$$

where $G_c(k, l, m)$ denotes the probability that A -voters win if there are k A -voters, l B -voters and m delegators who delegate their votes uniformly, with an individual voter being allowed to have c votes at most. If one party has an excess of delegated votes then the excess votes go to the other party. For example, if h out of m votes are delegated to k A -voters and $k+h > ck$ then B -voters receive additional $k+h-ck$ votes, given that $l+m-h < cl$. If both parties reach the cap, all further excess votes are destroyed. We distinguish between four cases when h votes are delegated to k A -voters and $m-h$ votes are delegated to l B -voters:

- **Case a :** $k+h > ck$;
- **Case a' :** $k+h \leq ck$;
- **Case b :** $i-k+m-h > c(i-k)$;
- **Case b' :** $i-k+m-h \leq c(i-k)$.

We have to consider the four cases: ab , $a'b'$, ab' and $a'b$.

We define $G_c(k, l, m)$ in the following way:

$$G_c(k, l, m) = \begin{cases} \frac{1}{2} & \text{if } k=l=0, \\ \sum_{h=0}^m \binom{m}{h} \binom{k}{k+l}^h \binom{l}{k+l}^{m-h} r(k, l, h) & \text{else.} \end{cases}$$

where

$$r(k, l, h) = \begin{cases} g(ck, cl) & \text{in Case } ab, \\ g(k+h, l+m-h) & \text{in Case } a'b', \\ g(k+l+m-cl, cl) & \text{in Case } ab', \\ g(ck, l+m+k-ck) & \text{in Case } a'b. \end{cases}$$

For notational convenience, we denote a binomial random variable with parameters n and p by $\text{Bin}(n, p)$, for any $n \in \mathbb{N}$ and $p \in [0, 1]$.

4 Results

To gain an intuition for the formal results, we start with numerical examples. For this, we model the number of voters as a Poisson random variable, with average n . That is, $F(i) = \frac{n^i}{e^n i!}$. To indicate that we are considering a specific distribution with a parameter, we include the parameter in the function $P(\cdot)$. Thus, if we consider a Poisson distribution with parameter n , we write $P(n, p)$ instead of $P(p)$ and $P(n, p, m)$ instead of $P(p, m)$.

This assumption is a standard tool rendering the analysis of voting outcomes analytically tractable. Namely, by the *decomposition property*, the number of A -

voters is a Poisson random variable with average $n \cdot p$ and the number of B -voters is a Poisson random variable with average $n \cdot (1 - p)$. Moreover, these two random variables are independent.⁵

Table 1 reveals that the likelihood of A winning is smaller with free delegation than under conventional voting. The same occurs with capped delegation but in a slightly less pronounced way than with free delegation. We also observe that when m increases, both the winning probabilities for free delegation and for capped delegation first decline and then start to converge to the same probability as under conventional voting. The pattern in Table 1 repeats for different values of n and p . In the following, we provide formal results.

4.1 Free Delegation

We start with free delegation and show the following:

Theorem 1 $P(p, m) < P(p)$ for any $m \geq 1$ and $p > 0.5$.

In other words, Theorem 1 says that free delegation probabilistically favors the ex-ante minority, since the probability that the minority wins under vote delegation is $1 - P(p, m)$, while the probability that the minority wins in conventional voting is equal to $1 - P(p)$. A trivial observation is that if $m = 0$, then both $P(p, m)$ and $P(p)$ coincide, since there is no vote to delegate.

We note that our result rests on the assumption that types (preferences) of voters are stochastically independent. By a counterexample, we illustrate that delegation may favor the majority in alternative uncertainty models. Suppose, for instance, that there exists a joint distribution J over the pair of natural numbers (a, b) , where a denotes the number of voters for alternative A and b denotes the number of voters for alternative B . Suppose the probability that there are more A -voters than B -voters is strictly larger than $\frac{1}{2}$. We next construct a distribution J such that the probability that alternative A is winning increases under delegation.

Table 1 Probabilities of A winning under conventional voting $P(n, p)$, free delegation $P(n, p, m)$ and capped delegation $P_c(n, p, m)$ for cap $c = 2$

n	p	m	$P(n, p)$	$P(n, p, m)$	$P_2(n, p, m)$
20	0.6	1	0.81413	0.808443	0.808443
20	0.6	2	0.81413	0.804256	0.804256
20	0.6	5	0.81413	0.796578	0.796616
20	0.6	10	0.81413	0.791246	0.792627
20	0.6	300	0.81413	0.808516	0.81413

⁵ Poisson games were introduced by Myerson [11].

Example 1 We assume that the joint distribution J is the product of two distribution functions f and g , i.e. $J(a, b) = f(a) \cdot g(b)$. Suppose that $f(4) = 0.6$, $f(1) = 0.4$ and $g(2) = 1$. Then $J(4, 1) = 0.6$ and $J(1, 2) = 0.4$ and $m = 1$. The probability that alternative A wins under conventional voting when the delegator abstains is 0.6. The probability that alternative A wins under delegation is⁶

$$0.6 + f(1) \cdot \frac{1}{2} \cdot \frac{1}{3} > 0.6.$$

4.2 Capped Delegation

We establish the following results in the case of capped delegation:

Theorem 2 $P_c(p, m) < P(p)$ for any $m \geq 1$, $c > 1$ and $p > 0.5$.

Theorem 2 shows that capped delegation probabilistically favors the ex-ante minority. We note that if $c = 1$ or $m = 0$, $P_c(p, m)$ and $P(p)$ coincide, since every voter can have at most one vote or no vote is delegated.

Conjecture 1 $P(p, m) \leq P_c(p, m)$ for any $m \geq 1$, $c \geq 1$ and $p > 0.5$.

Conjecture 1 shows that capped delegation is better for the ex-ante majority than free delegation. Together with Theorem 2, it implies that capped delegation is in-between standard voting (no delegation) and free delegation with respect to the probability that the majority wins. Note that if $c \geq m + 1$, then $P(p, m)$ and $P_c(p, m)$ coincide, since the cap is higher than the total amount of delegated votes. For $c = 1$, we have strict inequality following from Theorem 1 because $P_1(p, m) = P(p)$. Finally, for $m = 1$ and $c > 1$, again, $P(p, m)$ and $P_c(p, m)$ coincide. Numerical calculations support the statement of the conjecture for the remaining cases of c and m , see e.g., Table 1.

4.3 Asymptotic Behavior of Delegation

In this section, we consider three cases of asymptotic behavior of delegation. In the first, the probability that a voter belongs to the majority is large. In the second, the average total number of voters is large. Both cases describe typical assumptions in real-world situations. In the first, one assumes there are very few voters in the minority, such as, for example, is the case with malicious voters in blockchains. In the second, we look at a large electorate, with fixed shares of majority and minority voters. In the third case, we show the convergence of free delegation towards conventional voting as the number of delegators goes to infinity.

⁶ One can construct examples in which the increase of the likelihood that the majority wins is as high as 0.25 in the limit.

First, we obtain a rather straightforward proposition:

Proposition 1 *For any fixed natural number $m \geq 1$, we have $\lim_{p \rightarrow 1} P(p, m) = 1$.*

Proposition 1 shows that for any number of delegators, the probability that alternative A wins is 1 when the probability that a voter is an A -voter goes to 1. Next, we show that the probability that A wins with free delegation converges to the probability that A wins with conventional voting if m goes to infinity.

Proposition 2 $\lim_{m \rightarrow \infty} P(p, m) = P(p)$ for any $p \in (0, 1)$.

Another direct corollary of Proposition 2 and Conjecture 1 for capped delegation is that $\lim_{m \rightarrow \infty} P_c(p, m) = P(p)$ for any $c \in \mathbb{N}$ and $p \in (0, 1)$.

At the end of this section, we consider a large electorate. Large elections are modeled using a Poisson random variable with parameter n . It is straightforward that with a constant number of delegators, the probability that the majority wins converges to one as the total population size converges to infinity. We show that the same holds even if there are arbitrarily many delegators. In particular, the result holds if the number of delegators is a function of n . The latter is the “bottleneck” of the following proposition:

Proposition 3 $\lim_{n \rightarrow \infty} P(n, p, m) = 1$ for any fixed $p > 0.5$ and any m , where m can even depend on n .

This result can be generalized to other distributions F , by suitably defining the corresponding value of n . A direct corollary of Proposition 3 and Conjecture 1 with capped delegation is that $\lim_{n \rightarrow \infty} P_c(n, p, m) = 1$ for any fixed $p > 0.5$, any $c \in \mathbb{N}$ and any m , where m can even depend on n .

5 Discussion and Conclusion

We showed that the introduction of vote delegation leads to lower probabilities of winning for the ex-ante majority. Numerical simulations show that free delegation leads to even lower winning probabilities than capped delegation. Both delegation processes lead to lower probabilities that the majority wins than conventional voting. These results are particularly important in blockchain governance if one wants to prevent dishonest agents from increasing their probability of winning. Although the setting we analyze in the paper is as simple as possible, we already obtain non-trivial observations.

Our results are interesting in the context of delegation as a strategic decision. For the delegators, whether or not they belong to the ex-ante majority, choosing between delegating and abstaining becomes a strategic decision.

First, note that a setting in which no one delegates cannot be an equilibrium state, because minority voters prefer to delegate. Once some of them delegate, the probability that the majority of voters is winning is smaller than in conventional

voting. We conjecture that with a large number of delegators, a setting in which all majority voters delegate is sustainable in the equilibrium if the winning probability is increasing, starting at some threshold. We leave this challenging issue to future research.

References

1. Abramowitz, B., & Mattei, N. (2019). Flexible representative democracy: An introduction with binary issues. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization* (pp. 3–10). Macao, China: IJCAI-19. <https://doi.org/10.24963/ijcai.2019/1>.
2. Bloembergen, D., Grossi, D., & Lackner, M. (2019). On rational delegations in liquid democracy. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence* (pp. 1796–1803). Honolulu, Hawaii, USA: AAAI Press. <https://doi.org/10.1609/aaai.v33i01.33011796>.
3. Caragiannis, I., & Micha, E. (2019). A contribution to the critique of liquid democracy. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, International Joint Conferences on Artificial Intelligence Organization* (pp. 116–122). Macao, China: IJCAI-19. <https://doi.org/10.24963/ijcai.2019/17>.
4. Escoffier, B., Gilbert, H., & Pass-Lanneau, A. (2019). The convergence of iterative delegations in liquid democracy in a social network. In D. Fotakis & E. Markakis (Eds.), *Algorithmic Game Theory* (pp. 284–297). Cham: Springer.
5. Gersbach, H., Mamageishvili, A., & Schneider, M. (2021). Vote delegation and misbehavior. In I. Caragiannis, & K. A. Hansen (Eds.), *Algorithmic Game Theory—14th International Symposium* (pp. 21–24). Aarhus, Denmark: SAGT, Proceedings. Lecture Notes in Computer Science, vol. 12885, p. 411. Springer (2021).
6. Gözl, P., Kahng, A., Mackenzie, S., & Procaccia, A. D. (2018). The fluid mechanics of liquid democracy. In G. Christodoulou, & T. Harks (Eds.), *Web and Internet Economics—14th International Conference, WINE, Proceedings* (pp. 188–202). Oxford, United Kingdom. Lecture Notes in Computer Science, vol. 11316. Springer. https://doi.org/10.1007/978-3-030-04612-5_13.
7. Goodman, L. (2014). Tezos—A self-amending crypto-ledger. White Paper. <https://tezos.com/whitepaper.pdf>.
8. Kahng, A., Mackenzie, S., & Procaccia, A. (2018). Liquid democracy: An algorithmic perspective. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), 1095–1102. New Orleans, Louisiana, USA. <https://ojs.aaai.org/index.php/AAAI/article/view/11468>
9. Karakostas, D., Kiayias, A., & Larangeira, M. (2020). Account management in proof of stake ledgers. In C. Galdi & V. Kolesnikov (Eds.), *Security and Cryptography for Networks* (pp. 3–23). Cham: Springer.
10. Kotsialou, G., & Riley, L. (2020). Incentivising participation in liquid democracy with breadth-first delegation. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 638–644). Auckland, New Zealand: AAMAS '20, International Foundation for Autonomous Agents and Multiagent Systems.
11. Myerson, R. (1998). Population uncertainty and Poisson games. *International Journal of Game Theory*, 27, 375–392.
12. Pivato, M., & Soh, A. (2020). Weighted representative democracy. *Journal of Mathematical Economics*, 88, 52–63. <https://doi.org/10.1016/j.jmateco.2020.03.001>
13. Soh, A. (2020). Approval voting & majority judgment in weighted representative democracy. *SSRN Electronic Journal*. <https://doi.org/10.2139/ssrn.3731754>.

14. Wood, G. (2016). Polkadot: Vision for a heterogeneous multi-chain framework. <https://polkadot.network/PolkaDotPaper.pdf>.
15. Yakovenko, A. (2017). Solana: A new architecture for a high performance blockchain v0.8.13. <https://solana.com/solana-whitepaper.pdf>

Decentralized Governance of Stablecoins with Closed Form Valuation



Lucy Huo, Arian Klages-Mundt, Andreea Minca, Frederik Christian Munter, and Mads Rude Wind

Abstract We model incentive security in non-custodial stablecoins and derive conditions for participation in a stablecoin system across risk absorbers (vaults/CDPs) and holders of governance tokens. We apply option pricing theory to derive closed form solutions to the stakeholders' problems, and to value their positions within the capital structure of the stablecoin. We derive the optimal interest rate that is incentive compatible, as well as conditions for the existence of equilibria without governance attacks, and discuss implications for designing secure protocols.

Keywords Stablecoins · DeFi · Governance · Capital structure · Closed form valuation

1 Introduction

Decentralized finance (DeFi) protocols are often described as either utopian systems of aligned incentives or dystopian systems that incentivize hacks and exploits. These incentives, however, are thus far sparsely studied formally, especially around the governance of DeFi applications, which determine how they evolve over time. Unlike in traditional companies, governance in DeFi is meant to be transparent and openly auditable through smart contracts on a blockchain. The aim is often to incentivize good governance without relying on legal recourse, setting it apart from corporate finance. While some DeFi applications are immutable, with change impossible, most have some flexibility to parameters (such as fees and price feeds, or "oracles"), and often the entire functionality can be upgraded. Control is often placed in the hands of a cooperative of governance token holders who govern the system. This cooperative,

L. Huo · A. Klages-Mundt · A. Minca (✉)
Cornell University, New York, United States
e-mail: acm299@cornell.edu

F. C. Munter · M. R. Wind
Copenhagen University, Copenhagen, Denmark

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_4

however, is known to face perverse incentives, both theoretically (e.g., [7, 8, 22]) and often in practice (e.g., [6, 19]) to steal (or otherwise extract) value.

Related work. These incentives to deviate from the best interest of the protocol and its users are termed *governance extractable value* (GEV) [13, 21]. While there is a body of work on blockchain governance (e.g. [1, 12, 18]), DeFi governance is sparsely studied. [9] proposed a framework for modeling DeFi governance that extends capital structure models from corporate finance (see [5, 17]). Equilibria in these models are not yet formally studied. In this paper, we incorporate closed form valuation into the framework proposed in [9] and characterize the equilibria around interest rate policies (and how closely these lead to stability) and governance attacks in non-custodial stablecoins.

Non-custodial stablecoins. Non-custodial stablecoins are implemented as smart contracts using on-chain collateral, which are not controlled by a responsible party [3]. We briefly introduce the core components of these stablecoins and refer to [3, 9] for further details.

We focus on exogenous collateral, whose price is independent of the stablecoin system and which has proven able to maintain the peg in the long run, see e.g. [3]. Stablecoin issuance is initiated by a user creating a collateralized debt position (CDP) using a “vault”. The user transfers collateral, e.g., ETH, to the vault, which can mint an amount of stablecoins up to the minimum collateralization level. This leveraged position can be used in multiple ways, e.g. to spend the stablecoin or invest in other assets. The vault owner can redeem the collateral by reimbursing the vault with the issued stablecoins (with interest) and is tasked with maintaining the required collateralization.

If the vault becomes undercollateralized, for instance if the price of ETH drops, then an involuntary redemption (liquidation) is performed to deleverage the position. This deleveraging is performed through buy-backs of stablecoins to close the vault. Vaults are over-collateralized to help ensure that the position can be closed. However, should the liquidation proceeds be insufficient, additional mechanisms may kick in to cover the shortfall—either by tapping into a reserve fund or by selling governance tokens as a form of sponsored support (or backstop). Notably, this occurred in Dai on Black Thursday, when the Maker system found itself in a deleveraging spiral [10, 11, 14].

Incentive compatibility. Drawing from [21], we consider a cryptoeconomic protocol to be incentive compatible if “agents are incentivized to execute the game as intended by the protocol designer.” [9] reduces this to a key question of *incentive security*: Is equilibrium participation in the stablecoin sustainable? This requires that incentives among all participants (stablecoin holders, vault owners, governance agents) lead to a mutually profitable equilibrium of participating in the stablecoin. As non-custodial stablecoins contain self-governing aspects outside of most rule of law, participant incentives are also influenced by the possibility of profitable governance attacks.

This paper. We study governance incentive problems in non-custodial stablecoins similar to Maker. We formalize a game theoretic model (an adaptation of that in [9]) of governance incentives in Sect. 2. We derive closed form solutions to

stakeholders' problems in Sect. 3 using financial engineering methods, culminating in conditions for a unique equilibrium in Theorem 1. We then modify the model to include governance attacks in Sect. 4 and derive conditions for equilibria without attacks.

2 Model

We build upon the framework presented in [9], which seeks to describe incentives between governance token holders (GOV), vaults/risk absorbers and stablecoin holders. We define the model parameters in Table 1.

We first introduce the basic framework with no attack vectors. The setup considers an interaction between governors and vaults, who both seek to maximize expected profits. The governance choice problem is simply to maximize expected fee revenue. The vault choice problem is to maximize expected revenue from maintaining a long position in COL, while pursuing a new (leveraged) opportunity, and paying an interest fee to governance. Randomness is introduced in the model by assuming that the return on COL e^R follows a log-normal distribution:

$$R \sim N(0, \sigma^2),$$

where the standard deviation σ represents the COL volatility. We consider continuous compounding returns. The time horizon is set to 1. In this case $F(e^\delta - 1)$ represents the total interest paid by vault for the stablecoin issuance, while $FB(e^b - 1)$ represents the net revenue from investing the proceeds of the stablecoin issuance in the outside opportunity.

Further, vaults are subject to three constraints: Eq. (2), the collateral constraint, which restricts maximum stablecoin issuance to a fraction of posted collateral; Eq. (3), the stablecoin price, which is pegged at one whenever collateral does not fall short; and Eq. (4), the participation constraint, which simply states that

Table 1 Model components

Variable	Definition
N	Dollar value of vault collateral (COL position)
e^R	Return on COL
F	Total stablecoin issuance (debt face value)
b	Return rate on the outside opportunity
β	Collateral factor
δ	Interest rate paid by vault to issue STBL
u	Vault's utility from an outside COL opportunity
B	STBL market price

participation must yield higher utility compared to abandoning the system. Formally, the governance choice problem is written as:

$$\begin{aligned} \max_{\delta} \quad & \mathbb{E}[(e^{\delta} - 1) \cdot F] \\ \text{s.t.} \quad & \text{Vault's choice of } F. \end{aligned} \quad (1)$$

The vault choice problem can be written as:

$$\begin{aligned} \max_{F \geq 0} \quad & \mathbb{E} \left[\underbrace{Ne^R}_{\text{COL long position}} + \underbrace{F(B(e^b - 1) - (e^{\delta} - 1))}_{\text{Net revenue from leveraged position}} \right] \\ \text{s.t.} \quad & \text{GOV's choice of } \delta \\ & F \leq \beta N \end{aligned} \quad (2)$$

$$B = \mathbb{E} \left[\min \left(1, \frac{Ne^R}{F} - (e^{\delta} - 1) \right) \right] \quad (3)$$

$$u \leq \mathbb{E} [Ne^R + F(B(e^b - 1) - (e^{\delta} - 1))]. \quad (4)$$

We consider a Stackelberg equilibrium in which first the governance chooses an interest rate and then the vault chooses the stablecoin issuance. Note that the reverse order would yield a trivial solution in which the vault does not participate. The governance as a second player would simply set the interest rate $\delta \rightarrow \infty$. In anticipation, vault would set $F = 0$ as a consequence of the vaults' participation constraint. In contrast, the problem in which the governance moves first is non-trivial. Indeed, using financial engineering methods we will give closed form solutions to the objective functions of the two agents. This will allow us to analyze the convexity of their payoffs. Under reasonable conditions on parameters, the vaults' participation constraint imposes an upper bound on the interest rate but the optimal interest rate does not saturate this constraint.

Expected collateral shortfall. Our approach follows classical ideas for the valuation of corporate liabilities, present since the seminal works of Black and Scholes [2] and Merton [15, 16]. “Since almost all corporate liabilities can be viewed as combinations of options”, i.e., their payoffs can be replicated using an options portfolio, their valuations can be characterized using Black and Scholes formulas, see e.g., [20].

In analogy to the corporate debt holders, the stablecoin holders have an asset essentially equal to 1 (the face value) minus the following quantity that captures the collateral shortfall:

$$\begin{aligned} P(F, \delta) &= \mathbb{E} [Fe^{\delta} - Ne^R]_+ = Fe^{\delta} \Phi(-d_2) - N\Phi(-d_1) \\ d_1 &= \frac{\log(\frac{N}{Fe^{\delta}}) + \frac{\sigma^2}{2}}{\sigma}, \quad d_2 = d_1 - \sigma. \end{aligned} \quad (5)$$

Hence, the representative stablecoin holder mimicks the role of the debt holder in classical capital structure models. The quantity $\Phi(-d_1)$ represents the probability that there is a collateral shortfall (which is analogous to a corporate default in the classical corporate debt valuation theory): $F e^\delta > N e^R$. $\Phi(-d_2)$ is also the probability that there is a shortfall, but adjusted for the severity of this shortfall.¹

Vaults' Perspective In a similar manner, vaults take into account the expected collateral shortfall in their objective through the stablecoin price constraint,

$$\begin{aligned} \max_F \quad & N e^{\frac{\sigma^2}{2}} + F(e^b - e^\delta) - P(F, \delta)(e^b - 1) \\ \text{s.t.} \quad & F \leq \beta \cdot N \\ & u \leq N e^{\frac{\sigma^2}{2}} + F(e^b - e^\delta) - P(\delta, F)(e^b - 1) \\ & \text{GOV's choice of } \delta. \end{aligned} \tag{6}$$

GOV's Perspective Governance simply maximizes fee revenue

$$\begin{aligned} \max_\delta \quad & F(e^\delta - 1) \\ \text{s.t.} \quad & \text{Vault's choice of } F. \end{aligned} \tag{7}$$

We later consider an altered form of the model in Sect. 4 that incorporates a governance attack vector into our analysis.

3 Stackelberg Equilibrium Analysis

Governors know that the vaults will only choose to participate if the outside utility of an alternative COL usage is less than (or equal to) the benefit from issuing stablecoins. We first consider the optimum stablecoin issuance without the outside option constraint, and only with the leverage constraint. By evaluating vault objective sensitivities (see Appendix A), we can obtain vaults' objective maximizer if we include the leverage constraint (which imposes a cap on the amount of stablecoins issued) but ignore the participation constraint. All proofs are provided in Appendix B.

Proposition 1 *Vaults' unconstrained objective is maximized at*

$$\varphi(\delta) = N \cdot \exp \left[\sigma \cdot \Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right) - \delta - \frac{\sigma^2}{2} \right], \tag{8}$$

¹ Note that for the purposes of debt valuation the no-arbitrage theory of option pricing is not relevant. Only the Black and Scholes formulas are needed, i.e. the closed form solution for the expectation in (5) when the random return is log-normal. Moreover, while the valuation of corporate debt can be achieved in a dynamic model, the same formulas govern our one period case where the end of the period can be seen as the bond maturity.

which implicitly requires that $\delta \in [0, b]$. Moreover, if we include the leverage constraint, vaults' objective is maximized at

$$F^*(\delta) = \min(\varphi(\delta), \beta N). \quad (9)$$

By accounting for vaults' optimal issuance, GOV's objective transforms into

$$G = F^*(\delta) (e^\delta - 1). \quad (10)$$

3.1 $F^*(\delta)$ w/o Participation Constraint

We first derive results disregarding vaults' participation constraint. By comparing the unconstrained optimum stablecoin issuance $\varphi(\delta)$ to βN we pin down a lower bound δ_β for the interest rate, arising from the leverage constraint, such that $\forall \delta \in (\delta_\beta, b]$, $\varphi(\delta) < \beta N$. This result is due to vaults' preference for a larger stablecoin issuance when the interest rate is low, while being unable to exceed the leverage constraint set by governance. Along with the monotonicity of GOV's objective function, we obtain following proposition.

Proposition 2 (Governance choice) *There exists a $\delta_\beta \in [0, b]$ such that $\varphi(\delta_\beta) = \beta N$. GOV's optimal interest rate, δ^* , satisfies $\delta^* \geq \delta_\beta$.*

Hence, in equilibrium governance will either exhaust the leverage constraint with the highest compatible interest rate or set the interest rate higher than δ_β implying excess overcollateralization. This is not in itself surprising, yet leads us to the following technical lemma.

Lemma 1 (Concavity threshold for δ) *There exists a δ_{th} such that for $\delta > \delta_{th}$, GOV's objective is concave.*

The following Assumption 1 ensures that the volatility of the collateral's rate of return is not too large. With this, we further have that GOV's objective is locally increasing at $\delta = \delta_{th}$. This assumption is currently verified e.g. for ETH.

Assumption 1 $\sigma < 2\phi(0)$ (See Appendix for how this condition is derived).

By recalling that $\frac{dG}{d\delta}$ is non-increasing with δ for $\delta > \delta_{th}$ (from Lemma 1), we then obtain the following proposition.

Proposition 3 (Governance unconstrained optimal choice) *Under Assumption 1 there exists a δ^* , at which level GOV's objective is maximized. Consequently, if there exists $\delta_{th} < \delta_\beta < \delta^* \leq b$, then GOV would take δ^* .*

Then $\delta = \delta^*$ achieves the unconstrained maximum for the GOV objective. For $\delta_{th} < \delta_\beta$ to hold, we need the following assumption.

Assumption 2 $\beta < \frac{e^b+1}{2} \cdot \exp(-b - \frac{\sigma^2}{2})$.

This is because $\varphi(\delta_\beta) = \beta N$ and $\frac{d\varphi}{d\delta} < 0$, when we ask for $\delta_{th} < \delta_\beta$, we therefore need to have $\varphi(\delta_{th}) > \varphi(\delta_\beta) = \beta N$. Intuitively, governance will achieve a lower payoff at the interest rate pinned down by the leverage constraint, δ_β , relative to a lower interest rate, δ_{th} , which would, in turn, allow vaults to issue a larger amount of stablecoins resulting in larger interest revenue for governance ceteris paribus. Note that the RHS at Assumption 2 obtains its maximum value of $\exp(-\frac{\sigma^2}{2}) < 1$ when $b = 0$, implying overcollateralization.

3.2 $F^*(\delta)$ w/ Participation Constraint

We now give conditions on the parameters for which the optimal interest rate set by governors satisfies the vaults' participation constraint given outside utility on COL usage. First, we make the following additional assumption.

Assumption 3 $u \leq N e^{\frac{\sigma^2}{2}} + N \Phi(-d_1(\delta^*)) (e^b - 1)$.

Here we ensure that the vault is able to achieve a payoff equal to or above their utility from an outside COL opportunity. This assumption ensures that GOV is aware that their optimum interest rate must be sufficiently attractive in order for vaults to participate, i.e., governance must take into account the vaults' idiosyncratic tradeoffs. Armed with this assumption, we characterize a unique equilibrium in the following theorem.

Theorem 1 *If hyper-parameters are selected such that Assumption 1, 2, and 3 are satisfied, and there exist δ_β and δ^* that satisfy (24) and (28) respectively, then there is a unique equilibrium with $\delta = \delta^*$ and $F = \varphi(\delta^*)$.*

4 Governance Attack Vector

We now introduce a governance attack vector, as per [7]. A rational adversary only engages in an attack if its profits exceed costs. They could exploit the governance system to change the contract code and access a sufficiently large GOV token stake to approve the update. For instance, the adversarial change could transfer all COL to the adversary's address. More nuanced versions can also extract collateral indirectly by manipulating price feeds as in [8]. This may not require 50% of GOV tokens as governance participation is commonly low. Neither does it require a single wealthy adversary, since many attackers can collude via a crowdfunding strategy (e.g., [4]), or a single attacker could borrow the required tokens via a flash loan (as in [7, 22]). Note that timelocks make it harder to pursue flash loan governance attacks.

Formally, we consider an adversarial agent with a ζ fraction of GOV tokens, who is able to steal a γ fraction of collateral in the system. Typically, we might have

$\gamma = 1$, although not always. A rational attack will take place when profits exceed costs, i.e., when $\zeta F \frac{e^{\delta}-1}{1-r} + \alpha < \gamma \mathbb{E}[Ne^R] = \gamma Ne^{\frac{\sigma^2}{2}}$, where α is an outside cost to attack, and $\zeta F \frac{e^{\delta}-1}{1-r}$ is the opportunity cost of an attack, i.e. the profits resulting from a non-attack decision, represented as a geometric series of future fee revenue with discount factor r . In idealized DeFi, we might have $\alpha = 0$ or very close to 0 (through pseudonymity), while, in traditional finance, α is assumed to be so high such that an attack would always be unprofitable, e.g. due to legal repercussions.

We could extend the vault choice problem to include the amount of collateral, N , locked in the stablecoin system as a share of total endowed collateral available to the vault, \bar{N} . Only locked in collateral is subject to seizure during a governance attack. We assume for simplicity that $\bar{N} = N$ and we leave the decision on how much collateral to lock in as an open problem. If there is no attack, i.e., $\alpha + \zeta F \frac{e^{\delta}-1}{1-r} \geq \gamma Ne^{\frac{\sigma^2}{2}}$, the governance choice problem writes as before in (1) and if there is an attack, i.e., $\alpha + \zeta F \frac{e^{\delta}-1}{1-r} < \gamma Ne^{\frac{\sigma^2}{2}}$, then the governance's payoff (ex-adversary) is equal to zero.

In the Stackelberg equilibrium with vaults as a second player, the vault choice problem is only relevant conditional on the governance attack being unsuccessful, in which case it writes as before. If the attack is successful, then there is no participation from vaults and the stablecoin system is abandoned. We are thus interested in the non-attack scenario with participation from the vault, since only then is there mutually profitable continued participation for both parties and we have incentive security.

The optimal interest rate set by governance that ensures a non-attack decision (and so incentive security) then satisfies the following condition:

$$\alpha + \underbrace{\zeta F(\delta^*) \frac{e^{\delta^*}-1}{1-r}}_{G^*} \geq \gamma Ne^{\frac{\sigma^2}{2}} \quad \text{equivalently, } G^* \geq \frac{\gamma Ne^{\frac{\sigma^2}{2}} - \alpha}{\zeta}, \quad (11)$$

where G^* is the optimal governance objective value (and δ^* is the unique optimizing interest rate) under the assumptions of Theorem 1.

Since δ^* represents a Stackelberg equilibrium with vault participation, condition (11) is both necessary and sufficient for the existence of an interest rate that satisfies both the non-attack condition *and* the participation constraint. Note that an interest rate that satisfies condition (11) may not be feasible in general. The practical implication of the condition is that participants in the system can use it to verify the incentives of decentralized governors and whether given conditions lead to an equilibrium with incentive security or whether governors may have perverse incentives. The condition applies given *rational* behaviour, since agents know ex-ante if an attack will take place based on parameter values.

5 Conclusion

We have characterized the unique equilibrium arising in non-custodial stablecoins with decentralized governance. The payoff structure is based on closed form valuations of the positions of two stakeholders in the capital structure that underlies the stablecoin. We obtain the equilibrium interest rate and level of participation in settings without governance attacks (Theorem 1) and with a possible governance attack (Eq. 11). Using these closed form solutions, protocol designers can more easily account for the effects their design choices will have on economic equilibrium and incentive security in the system. Our results allow us to quantify how loose the participation constraint can be in order to allow governors to earn a sufficiently high profit in the stablecoin system such that it offsets the proceeds from attacking the system. The implication is that GOV tokens should be expensive enough (e.g., from fundamental value of ‘honest’ cash flows) so that it is unprofitable for outsiders to buy them with the sole purpose of attacking the system.

By comparing the precise value of the GOV tokens to the return of the collateral at stake, adjusted for the attack cost, we can evaluate the security and sustainability of decentralized governance systems. As the adjusted attack cost increases with α , one possible mitigation to strengthen these governance systems is the traditional one: increase α to deter attack through centralized means. One way to do this to make governors resemble legal fiduciaries with known identities, which often goes against the idealized tenets of DeFi. Another possibility, recently proposed in [13] as “optimistic approval”, alters the problem in a different way by incorporating a veto mechanism invocable by other parties in the system (e.g., vaults and stablecoin holders) in the case of malicious governance proposals. This would introduce a new term in our model that lowers the success probability of an attack based on the probability that the veto mechanism is invoked. If governors anticipate that the veto mechanism will be invoked, then their expectations of attack profit plummet, expanding the mutual participation region.

Acknowledgements The authors thank the Center for Blockchains and Electronic Markets at University of Copenhagen for support.

A Derivative Analysis

A.1 Sensitivity of the Expected Collateral Shortfall

Note the following relationship,

$$F e^{\delta} \phi(d_2) = N \phi(d_1) \quad (12)$$

With this, we have the following derivatives,

$$\begin{aligned}\frac{\partial P}{\partial F} &= e^\delta \Phi(-d_2) + F e^\delta \cdot \phi(-d_2) \cdot \left(-\frac{dd_2}{dF}\right) - N \cdot \phi(-d_1) \cdot \left(-\frac{dd_1}{dF}\right) \\ &= e^\delta \Phi(-d_2)\end{aligned}\quad (13)$$

$$\begin{aligned}\frac{\partial P}{\partial \delta} &= F e^\delta \Phi(-d_2) + F e^\delta \cdot \phi(-d_2) \cdot \left(-\frac{dd_2}{d\delta}\right) - N \cdot \phi(-d_1) \cdot \left(-\frac{dd_1}{d\delta}\right) \\ &= F e^\delta \Phi(-d_2)\end{aligned}\quad (14)$$

A.2 Vault Objective Sensitivities

Denote

$$V := N e^{\frac{\sigma^2}{2}} + F(e^b - e^\delta) - P(\delta, F)(e^b - 1)\quad (15)$$

Note the following derivatives,

$$\frac{\partial V}{\partial F} = (e^b - e^\delta) - (e^b - 1) e^\delta \Phi(-d_2)\quad (16)$$

$$\begin{aligned}\frac{\partial V}{\partial \delta} &= -F e^\delta - (e^b - 1) \frac{\partial P}{\partial \delta} \\ &= -F e^\delta - (e^b - 1) F e^\delta \Phi(-d_2) \\ &= -F e^\delta (1 + (e^b - 1) \delta \Phi(-d_2)) < 0 \quad \text{always}\end{aligned}\quad (17)$$

A.3 GOV Objective Sensitivities

Denote

$$G := F(e^\delta - 1)\quad (18)$$

Note the following derivatives,

$$\frac{\partial G}{\partial \delta} = F e^\delta > 0 \quad \text{always}\quad (19)$$

$$\frac{\partial G}{\partial F} = e^\delta - 1\quad (20)$$

B Proofs

Proposition 1

Proof Since V is concave in F , we set (16) equal to zero to obtain a maximum for V :

$$\begin{aligned}\Phi(-d_2) &= \frac{e^b - e^\delta}{e^\delta (e^b - 1)} \\ \frac{\log\left(\frac{F}{N}\right) + \delta + \frac{\sigma^2}{2}}{\sigma} &= \Phi^{-1}\left(\frac{e^{b-\delta} - 1}{e^b - 1}\right) \\ F^* = \varphi(\delta) &= N \cdot \exp\left[\sigma \cdot \Phi^{-1}\left(\frac{e^{b-\delta} - 1}{e^b - 1}\right) - \delta - \frac{\sigma^2}{2}\right],\end{aligned}\quad (21)$$

with (8) implicitly requiring that $\delta \in [0, b]$.

Together with the leverage constraint, this implies

$$F^*(\delta) = \min(\varphi(\delta), \beta N) \quad (22)$$

since the leverage constraint imposes a cap on amount of stablecoins issued. Substitute (9) into (18) and obtain

$$G = F^*(\delta) (e^\delta - 1), \quad (23)$$

thus transforming GOV's optimization into finding the optimum for (23).

Proposition 2

Proof We begin by establishing a lower bound for δ : There exists a $\delta_\beta \in [0, b]$ such that $\varphi(\delta_\beta) = \beta N$, i.e.

$$\frac{F^*}{N} = \exp\left[\sigma \cdot \Phi^{-1}\left(\frac{e^{b-\delta_\beta} - 1}{e^b - 1}\right) - \delta_\beta - \frac{\sigma^2}{2}\right] = \beta. \quad (24)$$

The quantity δ_β is the interest rate for which vaults' leverage constraint is hit, i.e. for $\delta < \delta_\beta$ the optimal stablecoin issuance is given by βN .

Indeed, by comparing $\varphi(\delta)$ to βN we obtain

$$\frac{d\varphi}{d\delta} = -\varphi(\delta) \cdot \left[1 + \sigma \cdot \frac{1}{\phi\left(\Phi^{-1}\left(\frac{e^{b-\delta}-1}{e^b-1}\right)\right)} \cdot \frac{e^{b-\delta}}{e^b-1}\right] < 0 \quad \text{always} \quad (25)$$

Thus $\exists \delta_\beta \in [0, b]$ such that $\varphi(\delta_\beta) = \beta N$, i.e.

$$\frac{F^*}{N} = \exp \left[\sigma \cdot \Phi^{-1} \left(\frac{e^{b-\delta_\beta} - 1}{e^b - 1} \right) - \delta_\beta - \frac{\sigma^2}{2} \right] = \beta, \quad (26)$$

which effectively is setting a lower bound to δ , such that $\forall \delta \in (\delta_\beta, b]$, $\varphi(\delta) < \beta N$.

We can no conclude the proof of Proposition 2. **Suppose** $\varphi(\delta) \geq \beta N$, i.e. $\delta \in [0, \delta_\beta]$

$$G = \beta N \cdot (e^\delta - 1)$$

$$\frac{dG}{d\delta} = \beta N e^\delta > 0 \quad \text{always.}$$

Thus, GOV will choose $\delta^* = \delta_\beta$.

Lemma 1

Proof Suppose $\varphi(\delta) < \beta N$, i.e. $\delta \in (\delta_\beta, b]$

$$G = \varphi(\delta) \cdot (e^\delta - 1)$$

$$\frac{dG}{d\delta} = \varphi(\delta) \cdot e^\delta + \frac{\partial \varphi}{\partial \delta} \cdot (e^\delta - 1)$$

$$= \varphi(\delta) \cdot e^\delta - \varphi(\delta) \cdot \left[1 + \sigma \cdot \frac{1}{\phi \left(\Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right) \right)} \cdot \frac{e^{b-\delta}}{e^b - 1} \right] \cdot (e^\delta - 1)$$

$$= \varphi(\delta) \left[1 - \sigma \cdot \frac{1}{\phi \left(\Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right) \right)} \cdot \frac{e^b - e^{b-\delta}}{e^b - 1} \right]$$

from which the lemma follows. Consider threshold value δ_{th} such that

$$\frac{e^{b-\delta_{\text{th}}} - 1}{e^b - 1} = 0.5 \quad \Rightarrow \quad \delta_{\text{th}} = b - \log \left(\frac{e^b + 1}{2} \right).$$

When $\delta > \delta_{\text{th}}$, we have as δ increases

- $\Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right)$ decreases from 0 to $-\infty$
- $\phi \left(\Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right) \right)$ hence decreases from $\phi(0)$ to 0
- $\frac{1}{\phi \left(\Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right) \right)}$ increases from $\frac{1}{\phi(0)}$ to ∞
- $\frac{e^b - e^{b-\delta}}{e^b - 1}$ increases from 0.5 to $\frac{e^b}{e^b - 1}$
- Overall, $\sigma \cdot \frac{1}{\phi \left(\Phi^{-1} \left(\frac{e^{b-\delta} - 1}{e^b - 1} \right) \right)} \cdot \frac{e^b - e^{b-\delta}}{e^b - 1}$ is increasing.

Assumption 1*Proof*

$$\begin{aligned}
& 1 - \sigma \cdot \frac{1}{\phi\left(\Phi^{-1}\left(\frac{e^{b-\delta_{th}}-1}{e^b-1}\right)\right)} \cdot \frac{e^b - e^{b-\delta_{th}}}{e^b - 1} > 0 \\
\Leftrightarrow & 1 - \frac{\sigma}{2\phi(0)} > 0 \\
\Leftrightarrow & \sigma < 2\phi(0)
\end{aligned} \tag{27}$$

Proposition 3

Proof Under Assumption 1, we have that G is locally increasing at $\delta = \delta_{th}$ and we have that $\frac{dG}{d\delta}$ is non-increasing with δ for $\delta > \delta_{th}$.

Therefore, when setting $\frac{dG}{d\delta} = 0$, i.e. $1 - \sigma \cdot \frac{1}{\phi\left(\Phi^{-1}\left(\frac{e^{b-\delta}-1}{e^b-1}\right)\right)} \cdot \frac{e^b - e^{b-\delta}}{e^b - 1} = 0$, we implicitly obtain a δ^* , at which level G is maximized.

Theorem 1*Proof* At δ^* , we have

$$\sigma \cdot \frac{1}{\phi\left(\Phi^{-1}\left(\frac{e^{b-\delta^*}-1}{e^b-1}\right)\right)} \cdot \frac{e^b - e^{b-\delta^*}}{e^b - 1} = 1 \tag{28}$$

Substitute (28) into (15),

$$\begin{aligned}
V(\delta^*) &= Ne^{\frac{\sigma^2}{2}} + \varphi(\delta^*)(e^b - e^{\delta^*}) \\
&\quad - P(\delta^*, \varphi(\delta^*))(e^b - 1) \\
P(\delta^*, \varphi(\delta^*)) &= \varphi(\delta^*)e^{\delta^*} \Phi(-d_2) - N\Phi(-d_1)
\end{aligned} \tag{29}$$

$$\begin{aligned}
d_1(\delta^*) &= \frac{1}{\delta} \cdot \left(\log\left(\frac{N}{\varphi(\delta^*)e^{\delta^*}}\right) + \frac{\sigma^2}{2} \right) \\
&= -\Phi^{-1}\left(\frac{e^{b-\delta^*}-1}{e^b-1}\right) + \sigma
\end{aligned}$$

$$d_2(\delta^*) = d_1 - \sigma = -\Phi^{-1}\left(\frac{e^{b-\delta^*}-1}{e^b-1}\right) \tag{30}$$

We obtain

$$P(\delta^*, \varphi(\delta^*)) = \varphi(\delta^*) \frac{e^b - e^{\delta^*}}{e^b - 1} - N\Phi(-d_1)$$

and

$$V(\delta^*) = Ne^{\frac{\sigma^2}{2}} + N\Phi(-d_1)(e^b - 1)$$

such that we must assume

$$u \leq Ne^{\frac{\sigma^2}{2}} + N\Phi(-d_1(\delta^*))(e^b - 1), \quad (31)$$

in order for vault participation to hold.

References

1. Beck, R., Müller-Bloch, C., & King, J. L. (2018). Governance in the blockchain economy: A framework and research agenda. *Journal of the Association for Information Systems*, 19(10), 1.
2. Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, 81(3), 637–654.
3. Bullmann, D., Klemm, J., & Pinna, A. (2019). In search for stability in crypto-assets: are stablecoins the solution? *ECB Occasional Paper*, 230.
4. Daian, P., Kell, T., Miers, I., & Juels, A. (2018). On-chain vote buying and the rise of dark DAOs. <https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/>.
5. Dybvig, P. H., & Zender, J. F. (1991). Capital structure and dividend irrelevance with asymmetric information. *The Review of Financial Studies*, 4(1), 201–219.
6. Foxley, W. (2020). \$10.8m stolen, developers implicated in alleged smart contract ‘rug pull’. *CoinDesk*. <https://www.coindesk.com/compounder-developers-implicated-alleged-smart-contract-rug-pull>.
7. Gudgeon, L., Perez, D., Harz, D., Livshits, B., & Gervais, A. (2020). The decentralized financial crisis. arXiv preprint [arXiv:2002.08099](https://arxiv.org/abs/2002.08099).
8. Klages-Mundt, A. (2019). Vulnerabilities in maker: oracle-governance attacks, attack daos, and (de)centralization. <https://link.medium.com/VZG64fmr6>.
9. Klages-Mundt, A., Harz, D., Gudgeon, L., Liu, J.Y., & Minca, A. (2020). Stablecoins 2.0: Economic foundations and risk-based models. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies* (pp. 59–79).
10. Klages-Mundt, A., & Minca, A. (2019). (in) stability for the blockchain: Deleveraging spirals and stablecoin attacks. arXiv preprint [arXiv:1906.02152](https://arxiv.org/abs/1906.02152).
11. Klages-Mundt, A., & Minca, A. (2020). While stability lasts: A stochastic model of stablecoins. arXiv preprint [arXiv:2004.01304](https://arxiv.org/abs/2004.01304).
12. Lee, B. E., Moroz, D. J., & Parkes, D. C. (2020). The political economy of blockchain governance. Available at SSRN 3537314.
13. Lee, L., & Klages-Mundt, A. (2021). Governance extractable value. <https://ournetwork.substack.com/p/our-network-deep-dive-2>.
14. MakerDAO (2020). Black thursday response thread. <https://forum.makerdao.com/t/black-thursday-response-thread/1433>.
15. Merton, R. C. (1970). A dynamic general equilibrium model of the asset market and its application to the pricing of the capital structure of the firm.
16. Merton, R. C. (1974). On the pricing of corporate debt: The risk structure of interest rates. *The Journal of finance*, 29(2), 449–470.
17. Myers, S. C., & Majluf, N. S. (1984). Corporate financing and investment decisions when firms have information that investors do not have. *Journal of financial economics*, 13(2), 187–221.

18. Reijers, W., O’Brocháin, F., & Haynes, P. (2016). Governance in blockchain technologies & social contract theories. *Ledger, 1*, 134–151.
19. Rekt. (2021). Paid network–rekt. <https://rekt.eth.link/paid-rekt/>.
20. Shreve, S. E., et al. (2004). Stochastic calculus for finance II: Continuous-time models, vol. 11. Springer.
21. Werner, S. M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., & Knottenbelt, W. J. (2021). Sok: Decentralized finance (defi). arXiv preprint [arXiv:2101.08778](https://arxiv.org/abs/2101.08778).
22. Zoltu, M. (2019). How to turn \$20m into \$340m in 15 seconds. <https://link.medium.com/k8QTaHzmr6>.

Griefing Factors and Evolutionary In-Stabilities in Blockchain Mining Games



Stefanos Leonardos, Shyam Sridhar, Yun Kuen Cheung,
and Georgios Piliouras

Abstract We revisit the standard game-theoretic model of blockchain mining and identify two sources of instabilities for its unique Nash equilibrium. In our first result, we show that *griefing*, a practice according to which participants of peer-to-peer networks harm other participants at some lesser cost to themselves, is a plausible threat that may lead cost-efficient miners to allocate more resources than predicted. The proof relies on the evaluation of *griefing factors*, ratios that measure network losses relative to an attacker's own losses and leads to a generalization of the notion of evolutionary stability to non-homogeneous populations which may be of independent game-theoretic interest. From a practical perspective, this finding provides explains the over-dissipation of mining resources, consolidation of power and high entry barriers that are currently observed in many mining networks. We, then, turn to the natural question of whether dynamic adjustments of mining allocations may, in fact, lead to the Nash equilibrium prediction. By studying two common learning rules, gradient ascent and best response dynamics, we provide evidence for the contrary. Thus, along with earlier results regarding in-protocol attacks, these findings paint a more complete picture about the various inherent instabilities of permissionless mining networks.

Keywords Blockchain mining · Evolutionary game theory · Griefing factors

S. Leonardos (✉)
King's College London, London, UK
e-mail: stefanos.leonardos@kcl.ac.uk

S. Sridhar · G. Piliouras
Singapore University of Technology and Design, Somapah Rd, Singapore
e-mail: shyam.sridhar@ethereum.org

G. Piliouras
e-mail: georgios@sutd.edu.sg

S. Sridhar
Ethereum Foundation, Bern, Switzerland

Y. K. Cheung
Royal Holloway, University of London, London, UK
e-mail: yunkuen.cheung@rhul.ac.uk

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes
in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_5

1 Introduction

A persisting barrier in the wider public adoption of permissionless blockchains is the uncertainty regarding their stability and long-term sustainability [4, 6]. The critical actors for the stability of the permissionless blockchains are the miners who provide their costly resources (e.g., computational power in Proof of Work (PoW) [17] or tokens of the native cryptocurrency in Proof of Stake (PoS) protocols) to secure consensus on the growth of the blockchain [5, 19, 30]. Miners act in self-interested, decentralized ways and may enter or leave these networks at any time. For their service, miners receive monetary rewards in return, typically in the form of transaction fees and newly minted coins of the native cryptocurrency in proportion to their individual resources in the network [8, 9]. The total amount of these resources, their distribution among miners, and the consistency over time in which they are provided are fundamental factors for the reliability of all blockchain supported applications [16, 31].

Theoretical models that study miners' incentives, typically use the Nash equilibrium prediction to argue about miner's behavior in an abstract way [3, 15]. However, empirical evidence suggests that miners' incentives are more intricate and this abstraction may lead to discrepancies between theoretical predictions and actual behavior observed in practice [13, 18]. For instance, miners may be willing to pay incur some (short-term) cost to harm (in some sense) their network competitors or, equivalently, to behave sub-optimally in the short run to gain an advantage in the long-run. Existing studies [40] and online resources that reflect investors' sentiment [33, 41], all suggest that the *allocation of mining resources in blockchain networks* remains poorly understood despite its critical role in the long-term success of the revolutionizing blockchain technology.

Model and contribution Motivated by the above, we study a standard game-theoretic model of blockchain mining [3]. According to this model, there exists a unique *Nash Equilibrium* (NE) allocation under the proportional reward scheme that is currently used by most Proof of Work (PoW) and Proof of Stake (Pos) protocols (Theorem 1). Having this as our starting point, we pose questions related to the stability properties of this NE from different perspectives that involve adversarial behavior at equilibrium and dynamic adjustment of resources over time. To answer these questions, we develop new tools to analyze miners' incentives and draw connections to existing concepts from game theory. Our results can be summarized as follows.

In our first set of results, we start with the observation that at the predicted NE levels, active miners are incentivised to deviate, by increasing their resources, in order to achieve higher *relative payoffs*. The loss that a deviating miner incurs to themselves is overcompensated by a larger market share and a higher loss that is incurred to each other individual miner and hence, to the rest of the network as a whole (Theorem 2, Corollary 1). While standing generally at odds with the metric of absolute payoffs, relative payoffs are particularly relevant in blockchain mining both due to the fixed supply of most cryptocurrencies (which incentivises miners to

maximize the amount of their tokens that they receive rather than their short-term profit) and due to adversarial behavior that is commonly observed in practice.

This practice, in which participants of a network cause harm to other participants, even at some cost to themselves, is known as *griefing*.¹ Our main technical insight is that griefing is closely related to the game-theoretic notion of *evolutionary stability*. Specifically, we quantify the effect of a miner's deviation via the (*individual*) *Griefing Factors (GF)*, defined as the ratios of network (or individual) losses over the deviator's own losses (Definition 2), and show that an allocation is *evolutionarily stable* if and only if all its individual GFs are less than 1 (Lemma 1).² We call such allocations (*individually*) *non-griefable* (Definition 3). This equivalence holds for homogeneous populations (i.e., for miners having equal mining costs) for which evolutionary stability is defined. However, as GFs are defined for arbitrary populations (not necessarily homogeneous), they provide a way to generalize evolutionary stable allocations as individually non-griefable allocations. Rephrased in this framework, our result states that the NE allocation is always individually griefable by miners who unilaterally increase their resources (Theorem 2).

In our second set of results, we, then, turn to the natural question of whether dynamic adjustments of mining allocations may, in fact, lead to the Nash equilibrium prediction. We show that learning (adjustment) rules that are commonly used to describe rational behavior in practice, such as Gradient Ascent (GA) and Best Response (BR) dynamics, exhibit chaotic or highly irregular behavior when applied in this game-theoretic model of blockchain mining (Sect. 5). This is true even for large number of miners and, in the case of GA, even for relatively small step-size (as long as miners are assumed to have influence via non-binding capacities on collective outcomes). Interestingly, these findings establish another source of instability of the equilibria of the game-theoretic model that is different in nature from the previous ones (algorithmic versus incentive driven).

Other Related Works Our paper contributes to the growing literature on miners' incentives in blockchain networks. By describing two previously unidentified sources of concern, griefing attacks and fluctuation of allocations when miners use standard adjustment rules, our findings complement existing results concerning inherent instabilities of mining protocols [11], manipulation of the difficulty adjustment in PoW protocols [15] or adversarial behavior [1, 22, 38]. Moreover, our results support the accumulating evidence that decentralization is threatened in permissionless blockchains [3, 26, 27] and offer an alternative rationale for the increased dissipation of resources (above optimal levels) that is observed in the main PoW mining networks [39]. A distinctive feature of this *over-mining* behavior in comparison to in-protocol adversarial behavior, is that it does not directly compromise the functionality of the blockchain. When a single miner increases their resources, the safety of

¹ The term *griefing* originated in multi-player games [42] and was recently introduced in blockchain related settings by [7].

² An allocation has a griefing factor of k if a miner can reduce others' payoffs by $\$k$ at a cost of $\$1$ to themselves by deviating to some other allocation.

the blockchain also increases. However, this practice has multiple negative effects as it generates a trend towards market concentration, dissipation of resources and high entry barriers. Finally, an important byproduct of our analysis is the formalization of the notion of grieving factors which is of independent interest in the context of the game-theoretic study of non-homogeneous populations in decentralized markets [12, 21, 32].

Outline Section 2 presents the strategic model (in a single blockchain) and studies the notions of grieving and evolutionary stability and Sect. 6 concludes the paper with a summary of our findings and some open questions.

2 Preliminaries

2.1 Model and Nash Equilibrium Allocations

We consider a network of $N = \{1, 2, \dots, n\}$ miners who allocate their resources, $x_i \geq 0$, to mine a blockchain-based cryptocurrency. Each miner $i \in N$ has an individual per unit cost $c_i > 0$. For instance, in Proof of Work (PoW) mining, x_i corresponds to TeraHashes per second (TH/s) and c_i to the associated costs (energy, amortized cost of hardware etc.) of producing a TH/s. We will write $\mathbf{x} = (x_i)_{i \in N}$ to denote the vector of allocated resources of all miners, and $X = \sum_{i=1}^n x_i$ to denote their sum. We will also write v to denote the total miners' revenue (coinbase transaction reward plus transaction fees) in a fixed time period (typically an epoch or a day in the current paper). The market share of each miner is proportional to their allocated resources (as is the case in most popular cryptocurrencies, see e.g., [7, 35]). Thus, the utility of each miner is equal to

$$u_i(x_i, \mathbf{x}_{-i}) = \frac{x_i}{x_i + X_{-i}} v - c_i x_i, \quad \text{for all } i \in N, \quad (1)$$

where, following standard conventions, we write $\mathbf{x}_{-i} = (x_j)_{j \neq i}$ and $X_{-i} := \sum_{j \neq i} x_j$ to denote the vector and the sum, respectively, of the allocated resources of all miners other than i . In Eq. (1), we may normalize v to 1 without loss of generality (by scaling each miner's utility by v). We will refer to the game, $\Gamma = (N, (u_i, c_i)_{i \in N})$, defined by the set of miners N , the utility functions u_i , $i \in N$ and the cost parameters c_i , $i \in N$ as the *mining game* Γ . As usual, a *Nash equilibrium* is a vector \mathbf{x}^* of allocations x_i^* , $i \in N$, such that

$$u_i(\mathbf{x}^*) \geq u_i(x_i, \mathbf{x}_{-i}^*), \quad \text{for all } x_i \neq x_i^*, \text{ for all miners } i \in N. \quad (2)$$

In terms of its Nash equilibrium, this game has been analyzed by [3]. To formulate the equilibrium result, let

$$c^* := \frac{1}{n-1} \sum_{i=1}^n c_i, \quad (3)$$

and assume for simplicity that $c^* > c_i$ for all $i \in N$. This is a *participation constraint* and implies that we consider only miners that are active in equilibrium. The unique Nash equilibrium of Γ is given in Theorem 1.

Theorem 1 ([3]) *At the unique pure strategy Nash equilibrium of the mining game Γ , miner $i \in N$ allocates resources $x_i^* = (1 - c_i/c^*)/c^*$. In particular, the total mining resources, X^* , allocated at equilibrium are equal to $X^* = 1/c^*$.*

Theorem 1 is our starting point. Our first task is to test the robustness of this Nash equilibrium in the context of decentralized and potentially adversarial networks. For instance, while the Nash equilibrium outcome is well-known to be incentive compatible, an adversary may decide to harm others by incurring a low(er) cost to himself. In decentralized networks, the (adversarial) practice of harming others at some lesser own loss is termed *griefing* [7]. As we show next, griefing is indeed possible in this case: a miner who increases their allocated resources above the Nash equilibrium prediction forgoes some of their own profits but incurs a (considerably) larger loss to the rest of the network. Our proof exploits a link between griefing and the fact that the Nash equilibrium is not evolutionary stable. To make these statements explicit, we first provide the relevant framework.

3 Evolutionary Stable Allocations and Griefing Factors

For this part, we restrict attention to homogeneous populations of miners, for which the notion of *evolutionary stability* is defined. Specifically, we consider a mining game $\Gamma = (N, (u_i, c_i)_{i \in N})$ such that all miners have equal costs, i.e., $c_i = c$ for some $c > 0$, for all $i \in N$. We will call such a game an *equal-cost mining game* and we will write $\Gamma = (N, c, u)$ for simplicity. A strategy profile $\mathbf{x} = (x_i)_{i \in N}$ is called *symmetric*, if there exists $x \geq 0$ such that $x_i = x$ for all $i \in N$. The following definition of evolutionary stability due to [20, 37] requires that $u_i(\mathbf{x}) = u_j(\mathbf{x})$ for any *symmetric* allocation $\mathbf{x} = (x)_{i \in N}$. This condition is trivially satisfied for equal-cost mining games, cf. Eq. (1).

Definition 1 (*Evolutionary Stable Allocation (ESA)*, [20, 37]) Let $\Gamma = (N, (u_i, c_i)_{i \in N})$ be a mining game such that $u_i \equiv u_j$ for all $i, j \in N$ for all symmetric allocation profiles $\mathbf{x} \geq 0$. Then, a symmetric allocation $\mathbf{x}^{\text{ESA}} = (x^{\text{ESA}})_{i \in N}$ is an *evolutionary stable allocation (ESA)* if

$$u_i(x_i, \mathbf{x}_{-i}^{\text{ESA}}) < u_j(x_i, \mathbf{x}_{-i}^{\text{ESA}}), \quad \text{for all } j \neq i \in N, x_i \neq x^{\text{ESA}}. \quad (4)$$

Definition 1 implies that an ESA, \mathbf{x}^{ESA} , maximizes the relative payoff function, $u_i(x_i, \mathbf{x}_{-i}^{\text{ESA}}) - u_j(x_i, \mathbf{x}_{-i}^{\text{ESA}})$ with $j \in N, j \neq i$, of any miner $i \in N$. Intuitively, if all

miners select an ESA, then there is no other allocation that could give an individually deviating miner a higher *relative payoff*. In other words, if a symmetric allocation $x_i = x, i \in N$, is not ESA, then there exists a $x' \neq x$, so that a single miner who deviates to x' has a strictly higher payoff (against x of the other $n - 1$ miners) than every other miner who allocates x (against $n - 2$ other miners who allocate x and the deviator who allocates x') [20].

As mentioned above, evolutionary stability is defined for homogeneous populations and may be, thus, of limited applicability for practical purposes. To study non-homogeneous populations, we will need a proper generalization of evolutionary stability. To achieve this, we introduce the notion of *griefing factors* which, as we show, can be used to formulate evolutionary stability and which is readily generalizable to arbitrary settings. This is done next.

Definition 2 (*Griefing Factors (GF)*) Let $\Gamma = (N, (u_i, c_i)_{i \in N})$ be a mining game (not necessarily of equal-costs) in which all miners are using the allocations $x_i^*, i \in N$, and suppose that a miner i deviates to an allocation $x_i \neq x_i^*$. Then, the *griefing factor*, (GF), of strategy x_i with respect to strategy x^* is defined by

$$\begin{aligned} \text{GF}_i((x_i, \mathbf{x}_{-i}^*); \mathbf{x}^*) &:= \frac{\text{loss incurred to the network}}{\text{deviator's own loss}} \\ &= \frac{\sum_{j \neq i}^n [u_j(\mathbf{x}^*) - u_j(x_i, \mathbf{x}_{-i}^*)]}{u_i(\mathbf{x}^*) - u_i(x_i, \mathbf{x}_{-i}^*)}, \end{aligned} \quad (5)$$

for all $i \in N$, where *loss* is the same as *utility loss*. The GF with respect to an allocation x^* can be then defined as the supremum over all possible deviations, i.e.,

$$\text{GF}(\mathbf{x}^*) = \sup_{i \in N, x_i \geq 0} \{ \text{GF}_i((x_i, \mathbf{x}_{-i}^*); \mathbf{x}^*) \}.$$

We can also define the *individual griefing factor of strategy x_i with respect to strategy x^* against a specific miner j* , as follows

$$\begin{aligned} \text{GF}_{ij}((x_i, \mathbf{x}_{-i}^*); \mathbf{x}^*) &:= \frac{\text{loss incurred to miner } j}{\text{deviator's own loss}} \\ &= \frac{u_j(\mathbf{x}^*) - u_j(x_i, \mathbf{x}_{-i}^*)}{u_i(\mathbf{x}^*) - u_i(x_i, \mathbf{x}_{-i}^*)} \end{aligned} \quad (6)$$

for all $j \neq i \in N$, where, as in Eq. (5), *loss* is a shorthand for *utility loss*. It holds that $\text{GF}_i((x_i, \mathbf{x}_{-i}^*); \mathbf{x}^*) = \sum_{j \neq i} \text{GF}_{ij}((x_i, \mathbf{x}_{-i}^*); \mathbf{x}^*)$.

As mentioned in Definition 2, the numerator of GF corresponds to the loss of all miners other than i incurred by i 's deviation to x_i , whereas the denominator corresponds to miner i 's own loss (cf. Eq. (5)). In decentralized mechanisms (e.g., blockchains), this metric captures an important *incentive compatibility* condition: namely, a mechanism is safe against manipulation if the costs of an attack exceed its potential benefits

to the attacker. This motivates to define an allocation as *griefable* if its GF is larger than 1.

Definition 3 (*Griefable and Individually Griefable Allocations*) An allocation $\mathbf{x}^* = (x_i^*)_{i \in N}$ is *griefable* if $\text{GF}(\mathbf{x}^*) > 1$. An allocation $\mathbf{x}^* = (x_i^*)_{i \in N}$ is *individually griefable* if there exist $i, j \in N$ and $x_i \neq x_i^* \geq 0$, such that the individual griefing factor $\text{GF}_{ij}((x_i, \mathbf{x}_{-i}^*); \mathbf{x}^*)$ is larger than 1.

An important observation is that the condition of evolutionary stability can be expressed in terms of the individual griefing factors. In particular, an allocation \mathbf{x}^{ESA} is evolutionary stable if and only if all *individual* griefing factors are less than 1, i.e., if and only if \mathbf{x}^{ESA} is not individually griefable. This is formalized in Lemma 1.

Lemma 1 *Let $\Gamma = (N, c, u)$ be an equal-cost mining game. Then, an allocation $\mathbf{x}^{\text{ESA}} = (x^{\text{ESA}})_{i \in N}$ is evolutionary stable if and only if \mathbf{x}^{ESA} is not griefable, i.e., iff*

$$\text{GF}_{ij}((x_i, \mathbf{x}_{-i}^{\text{ESA}}); \mathbf{x}^{\text{ESA}}) < 1, \quad \text{for all } j \neq i \in N, x_i \neq x^{\text{ESA}}. \quad (7)$$

Proof Since $u_i \equiv u_j$ for all symmetric \mathbf{x} and all $i, j \in N$ by assumption, we may write Eq. (4) as

$$u_i(x', \mathbf{x}_{-i}^{\text{ESA}}) - u_i(\mathbf{x}^{\text{ESA}}) < u_j(x', \mathbf{x}_{-i}^{\text{ESA}}) - u_j(\mathbf{x}^{\text{ESA}}),$$

for all $j \neq i \in N$ and for all $x' \neq x^{\text{ESA}}$. Since $u_i(x', \mathbf{x}_{-i}^{\text{ESA}}) < u_i(\mathbf{x}^{\text{ESA}})$ for all $x \neq x^{\text{ESA}}$ and for any miner $i \in N$, we may rewrite the previous equation as

$$1 > \frac{u_j(x', \mathbf{x}_{-i}^{\text{ESA}}) - u_j(\mathbf{x}^{\text{ESA}})}{u_i(x', \mathbf{x}_{-i}^{\text{ESA}}) - u_i(\mathbf{x}^{\text{ESA}})} = \text{GF}_{ij}((x', \mathbf{x}_{-i}^{\text{ESA}}); \mathbf{x}^{\text{ESA}}),$$

for all $j \neq i \in N$ and for all $x' \neq x^{\text{ESA}}$. This proves the claim. \square

Thus, Lemma 1 suggests that an allocation is evolutionary stable if and only if it is individually non-griefable. According to Definition 3, this is weaker than an allocation being non-griefable, which is satisfied if for all $i \in N$, the sum over $j \neq i \in N$ of all individual griefing factors G_{ij} is less than 1.

4 Griefing in Blockchain Mining

Lemma 1 suggests a way to extend the notion of evolutionary stability beyond homogeneous populations. In particular, for general, non-homogeneous populations, we may impose the stability requirement that an allocation should be individually non-griefable or, as mentioned above, the stronger requirement that an allocation should be non-griefable. This brings us to the main result of this section, which suggests that the Nash equilibrium of Theorem 1 is griefable for both homogeneous (symmetric)

and non-homogeneous (asymmetric) populations of miners. In particular, assuming that the network has stabilized at the x^* equilibrium allocation, a strategic miner may attack other miners simply by increasing their own mining resources. Specifically, if a miner i deviates to a resource allocation $x_i^* + \Delta$ for some $\Delta > 0$, then this creates a GF equal to $\mathcal{O}(n/\Delta)$. Such a deviation reduces the attacking miner's own payoff but, as we will see, it decreases the payoff of all other miners by a larger margin. This improves the attacking miner's *relative payoff* and hence their long-term survival chances in the blockchain mining network. This is formalized in Theorem 2.

Theorem 2 *Let $\Gamma = (N, (u_i, c_i)_{i \in N})$ be a mining game and let $\mathbf{x}^* = (x_i^*)_{i \in N}$ be its unique pure strategy Nash equilibrium.*

- (i) *In a homogeneous population, i.e., when all miners have the same cost, $c_i = c > 0$ for all $i \in N$, the unique Nash equilibrium allocation $x^* = \frac{n-1}{n^2c}$ is not evolutionary stable. In particular, there exists $x' \neq x^*$, so that an individually deviating miner i increases their relative payoff $u_i(x', x_{-i}^*) - u_i(x^*, x_{-i}^*)$.*
- (ii) *In a general, non-homogeneous population, the pure Nash equilibrium x^* is grieveable. In particular, assuming that all miners $j \in N$ are using their equilibrium allocations x_j^* , $j \in N$, the deviation $x_i^* + \Delta$, for some $\Delta > 0$, of miner $i \in N$, has a grieving factor*

$$\text{GF}_i((x_i^* + \Delta, \mathbf{x}_{-i}^*); \mathbf{x}^*) = \frac{n-1}{\Delta \cdot \sum_{j=1}^n c_j} = \mathcal{O}(n/\Delta).$$

In particular, at the Nash equilibrium allocation, x^ , any single miner may increase their mining resources and improve their utility in relative terms.*

- (iii) *In both the homogeneous and non-homogeneous populations, the unique individually non-grieveable allocation, $\mathbf{y} = (y_i)_{i \in N}$, satisfies $y_i = \frac{n}{n-1}x_i^*$, where x_i^* is the Nash equilibrium allocation of miner $i \in N$.*

Proof See Appendix A. □

Remark 1 Part (ii) of Theorem 2 reveals one shortcoming of the current definition of GF. Specifically, the GF may grow arbitrarily large as $\Delta \rightarrow 0$. However, as $\Delta \rightarrow 0$, the *absolute* total harm to the network is negligible (even if the relative loss is very large as expressed by the GF). One possibility to circumvent this problem is to consider discrete increments for Δ , i.e., $\Delta \in \{1, 2, \dots, 100, \dots\}$ as in e.g., [9]. Alternatively, one may combine GF with the absolute loss of the network to obtain a more reliable measure. We do not go deeper into this question at the current moment since it seems to be better suited for a standalone discussion. We leave this analysis as an intriguing direction for future work.

Remark 2 Part (iii) of Theorem 2 allows us to reason about the overall expenditure at the unique individually non-grieveable allocation $\mathbf{y} = (y_i)_{i \in N}$. In the general case, that of a non-homogeneous population, the total expenditure at an individually non-grieveable allocation $\mathbf{y} = (y_i)_{i \in N}$ is

$$E(\mathbf{y}) = \sum_{i \in N} c_i y_i = \frac{n-1}{n} \sum_{i \in N} c_i x_i^* = n \left[1 - (n-1) \frac{\sum_i c_i^2}{(\sum_i c_i)^2} \right],$$

where we used that $x_i^* = (1 - c_i/c^*)/c^*$ and $y_i = \frac{n}{n-1}x^*$ by Theorem 1 and part (iii) of Theorem 2, respectively. Cauchy-Schwarz inequality implies that $(\sum_i c_i)^2 \leq n \sum_i c_i^2$ which yields that $E(\mathbf{y}) \leq 1$ with equality if and only if $c_i = c$ for all $i \in N$. Thus, the expenditure in the individually non-grieffable allocation is always less than or equal to the aggregate revenue generated by mining, with equality only if the population is homogeneous. In that case, i.e., if all miners have the same cost $c_i = c$ for all $i \in N$, then the unique individually non-grieffable allocation is also evolutionary stable (cf. Lemma 1), i.e., $\mathbf{y} = \mathbf{x}^{\text{ESA}}$ with $x^{\text{ESA}} = \frac{1}{nc}$ for all $i \in N$ (by part (iii) and symmetry). In all cases, the total expenditure $E(\mathbf{x}^*)$, at the unique Nash equilibrium \mathbf{x}^* must be equal to $E(\mathbf{x}^*) = \frac{n-1}{n}E(\mathbf{y})$ and hence it less than the expenditure at the unique individually non-grieffable allocation and strictly less than the generated revenue (which is equal to 1). Finally, note that in practice, griefing may occur for via several other ways (e.g., direct sabotage, censoring etc. , so non-grieffable here should be interpreted only in the specific context of allocating more (or less) resources.

In the proof of Theorem 2, we have actually shown something slightly stronger. Namely, miner i 's individual loss due to its own deviation to $x_i^* + \Delta$ is less than the loss of each other miner j provided that Δ is not too large. In other words, the individual griefing factors with respect to the Nash equilibrium allocation are all larger than 1 and hence, the Nash equilibrium is also individually grieffable. This is formalized next.

Corollary 1 *For every miner $j \in N$ such that $\Delta < x_j^*$, it holds that*

$$\text{GF}_{ij}((x_i^* + \Delta, \mathbf{x}_{-i}^*); \mathbf{x}^*) > 1,$$

i.e., the loss of miner j is larger than the individual loss of miner i .

Proof of Corollary 1 By Eqs. (8) and (9), the inequality

$$u_j(\mathbf{x}^*) - u_j(x_i^* + \Delta, \mathbf{x}_{-i}^*) > u_i(\mathbf{x}^*) - u_i(x_i^* + \Delta, \mathbf{x}_{-i}^*)$$

is equivalent to

$$\begin{aligned} \frac{\Delta(c^* - c_j)}{1 + c^*\Delta} > \frac{\Delta^2 c_i c^*}{1 + c^*\Delta} &\iff c^* - c_j > \Delta c_i c^* \\ &\iff \Delta < \frac{1}{c_i} \left(1 - \frac{c_j}{c^*}\right). \end{aligned}$$

Since $c_i < c^*$ by assumption, and $x_j^* = \frac{1}{c^*} \left(1 - \frac{c_j}{c^*}\right)$ by Theorem 1, the right hand side of the last inequality satisfies

$$\frac{1}{c_i} \left(1 - \frac{c_j}{c^*}\right) > \frac{1}{c^*} \left(1 - \frac{c_j}{c^*}\right) = x_j^*.$$

This implies that $\Delta < x_j^*$ is sufficient for the initial inequality to hold which concludes the proof. \square

Theorem 2 and Corollary 1 imply that miners are incentivised to exert higher efforts than the Nash equilibrium predictions. The effect of this strategy is twofold: it increases their own relative market share (hence, their long-term payoffs) and harms other miners. The notable feature of this *over-mining* attack (or deviation from equilibrium) is that it does not undermine the protocol functionality directly. As miners increase their *constructive effort* to, security of the blockchain network also increases. This differentiates the blockchain paradigm from conventional contests in which griefing occurs via exclusively destructive effort or deliberate sabotage against others [2, 25].

However, the over-mining strategy has implicit undesirable effects. As we show next, it leads to consolidation of power by rendering mining unprofitable for miners who would otherwise remain active at the Nash equilibrium and by raising entry barriers for prospective miners. This undermines the (intended) decentralized nature of the blockchain networks and creates long-term risks for its sustainability as a distributed economy. Again, this is a distinctive feature of decentralized, blockchain-based economies: for the security of the blockchain to increase, it is desirable that the (honest) aggregate resources *and* their distribution among miners both increase (which is not the case in the over-mining scenario).

Proposition 1 *Let $\Gamma = (N, (u_i, c_i)_{i \in N})$ be a mining game with unique Nash equilibrium allocation $\mathbf{x}^* = (x_i^*)_{i \in N}$. Assume that all miners $j \neq i \in N$ are allocating their equilibrium resources x_j^* , and that miner i allocates $x_i^* + \Delta$ resources for some $\Delta > 0$. Then*

- (i) *the maximum increase Δ_i of miner i before miner i 's payoff becomes zero is $\Delta_i = \frac{1}{c_i} - \frac{1}{c^*}$.*
- (ii) *the absolute losses of all other miners $j \neq i$ are maximized when $\Delta = \Delta_i$ and are equal to $c_j x_i^*$.*

Proposition 1 quantifies (i) the maximum possible increase, Δ_i , in the mining resources of a single miner before their profits hit the break-even point (i.e., become zero), and (ii) the absolute losses of all other miners when miner i increases their resources by some Δ up to Δ_i . As intuitively expected, more efficient miners can cause more harm to the network (part (i)) and in absolute terms, this loss can be up to the equilibrium spending $c_i x_i^*$ of miner i , assuming that miner i does not mine at a loss (part (ii)). While not surprising these findings provide a formal argument that cost asymmetries can be severely punished by more efficient miners and that efficient miners can grow in size leading ultimately to a centralized mining network.

Remark 3 In the current setting, the assumption of symmetric miners (miners with equal or at least almost equal cost) is less restrictive than it seems. The participation

constraint $c_i < c^* = \frac{1}{n-1} \sum_{j=1}^n c_j$ implies that the costs, c_i 's, of the active miners in equilibrium cannot be *too different*. This is formalized in Observation 1.

Observation 1 *Let $c_{\max} := \max_{i \in N} \{c_i\}$ denote the maximum mining cost among all active miners and let $\bar{c} = \frac{1}{n} \sum_{i=1}^n c_i$ denote the average mining cost. Then, the variance $\sigma_c^2 := \sum_{i=1}^n (c_i - \bar{c})^2$ of the per unit mining costs of all active miners in equilibrium satisfies*

$$\sigma_c^2 < c_{\max} \left(\frac{n}{n-1} - c_{\max} \right).$$

To gain some intuition about the order of magnitude of the bound derived in Observation 1 in real applications, we consider the BTC network. Currently, the cost to produce 1 TH/s consistently for a whole day is approximately equal to \$0.08. On the other hand, the total miners' revenue per day is in the order of magnitude of \$10million. Thus, in normalized units (as the ones that we work here), c_i would be equal to $c_i = 0.08/10m = \$8e - 09$.

5 Dynamic Adjustments of Mining Allocations

In the previous section, we saw that at the unique evolutionary stable equilibrium, griefing is not possible, however, at that equilibrium, miners fully dissipate the reward and cause further consolidation of power that raises entry barriers to prospective entrants. Thus, the system enters a positive feedback loop towards market concentration. Given these results, a question that naturally arises is whether typical learning dynamics converge to the equilibrium of the game-theoretic model.³ This would provide an argument in favor of the Nash equilibrium allocations and restore the metric of absolute payoffs as a driver of miners' behavior. However, by examining two of the most commonly-used adjustment rules that model rational human behavior, *Gradient Ascent* and *Best Response* dynamics [10], we provide a negative answer to this question.

Specifically, recall that the utility of miner i in the strategic (single blockchain) model is given by

$$u_i(x_i, X_{-i}) = \frac{x_i}{x_1 + X_{-i}} - c_i x_i, \text{ for all } x_i \geq 0$$

and $i = 1, 2, \dots, n$, where $X_{-i} = \sum_{j \neq i} x_j$ (cf. Eq. (1)). Thus, the *Gradient Ascent* (GA) update rule is given by

³ Apart from the allocation of mining resources over time, dynamic update rules have recently attracted widespread attention in blockchain economies due to the implementation of posted price mechanisms in the transaction fee market [14, 29, 34, 36].

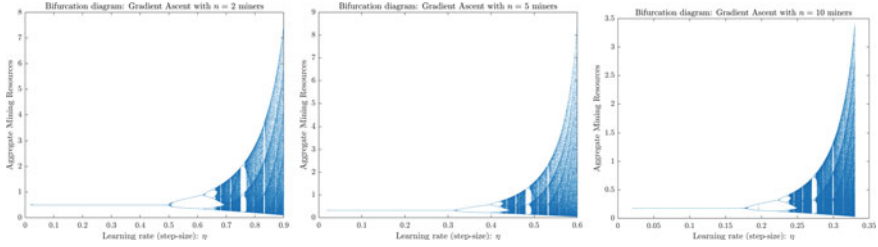


Fig. 1 Bifurcation diagrams for the Gradient Ascent dynamics with $n = 2, 5, 10$ miners with respect to the learning parameter θ . As the number of miners grows, the dynamics become chaotic for even lower step-sizes

$$x_i^{t+1} = x_i^t + \theta_i \frac{\partial}{\partial x_i^t} u_i(x_i^t) = x_i^t + \theta_i \left[\frac{X_{-i}^t}{(x_1^t + X_{-i}^t)^2} - c_i \right], \quad (\text{GA})$$

for all $i = 1, 2, \dots, n$, where θ_i is the learning rate of miner $i = 1, 2$. The bifurcation diagrams in Fig. 1 show the attractor of the dynamics for different values of the step-size (assumed here to be equal for all miners for expositional purposes) and for different numbers of active miners, $n = 2, 5$ and 10 , with $c_i = 1$ for all i . The blue dots show the aggregate allocated resources for 400 iterations after a burn-in period of 50 iterations (to ensure that the dynamics have reached the attractor).

All three plots indicate that the GA dynamics transition from convergence to chaos for relatively small values of the step-size. Interestingly, as the number of miners increases, the instabilities emerge for increasingly smaller step-size. This is in sharp contrast to the (PR) dynamics and their convergence to equilibrium under the large market assumption. The reason that a growing number of miners does not convey stability to the system is precisely because the miners are not assumed to have binding capacities. As miners act greedily, their joint actions drive the system to extreme fluctuations and the larger their number, the easier it is for these fluctuations to emerge. Finally, while convergence is theoretically established for small step-sizes in all cases, such step-sizes correspond to very slow adaption and are of lesser practical relevance.

We obtain a qualitatively similar result for the best response dynamics. The *Best Response (BR)* update rule is given by

$$x_i^{t+1} = \sqrt{X_{-i}^t / c_i} - X_{-i}^t, \quad \text{for all } i = 1, 2, \dots, n. \quad (\text{BR})$$

As above, the bifurcation diagrams in Fig. 2 show the aggregate allocated mining resources for $n = 2, 5$ and 10 miners. The horizontal axis (i.e., the bifurcation parameter) is now the cost asymmetry between the representative miner and all other miners which are assumed to have the same cost (again only for expositional purposes).⁴ The

⁴ Since PoW mining resembles an oligopoly [3], traditional economics suggest that cost asymmetries cannot be very large among *active* miners [28].

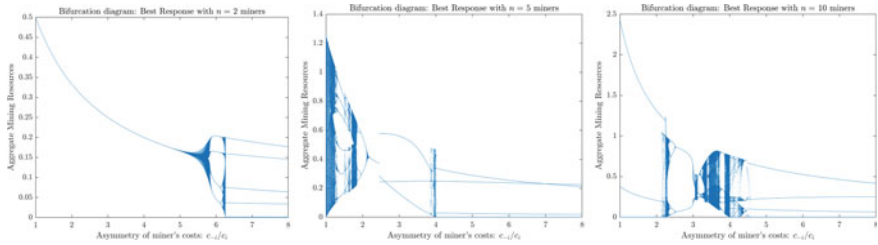


Fig. 2 Bifurcation diagrams for the Best Response dynamics with $n = 2, 5, 10$ miners with respect to the miners cost asymmetry. The dynamics become chaotic typically for intermediate values of cost asymmetry

plots suggest that the stability of the dynamics critically depend on the parameters of the system with chaos emerging for various configurations.

As showcased by the GA and BR dynamics, if miners’ decisions affect the decisions of other miners and if miners can adjust (increase or decrease) their capacities to optimize their profits, then common learning dynamics can exhibit arbitrary behavior. Instead of converging to the Nash equilibrium (or to some other stable outcome), the aggregate allocations may oscillate between extreme values or exhibit chaotic trajectories, with adverse effects on the reliability of the supported applications and the value of the blockchain-based cryptocurrency.⁵ Along with our earlier findings about griefing, these results paint a more complete picture about the various reasons that can destabilize permissionless blockchain networks when there is concentration of mining power.

6 Conclusions and Open Questions

In this paper, we studied resource allocation in blockchain mining. We identified two very different reasons for instabilities in mining networks: griefing, which involves the practice of harming others, even at some own cost and instability of dynamic allocation rules. Concerning the former, we showed that miners have incentives to increase their resources above the Nash equilibrium prediction, which explains the consolidation of power and increased entry barriers that are currently observed in practice. To do so, we measured the impact of griefing on miner’s payoffs via the *griefing factors*, i.e., ratios of network (or individual miner’s) losses relative to the attacker’s own losses [7], and showed that *non-griefable* states are precisely states that are evolutionary stable. As a byproduct, this provides a way to generalize the notion of evolutionary stability to non-homogeneous populations which may be of independent interest in game-theoretic models of decentralized settings. Concerning

⁵ The effects of blockchain related metrics, such as estimated hashrate, on cryptocurrency prices have been widely documented, see e.g., [23, 24] and the many references cited therein.

the latter, we provided evidence that rational learning rules such as Gradient Ascent and Best Response lead to chaotic allocations, or to the very least, allocations that changes significantly in response to small changes in the system parameters. Along with existing in-protocol attacks, such as selfish mining or manipulation of the difficulty adjustment in Proof of Work blockchains, these results paint a more complete picture of the inherent instabilities of permissionless mining networks.

Open questions A critical assumption of the game-theoretic model in which we derived the current results is that miners have large capacities of resources which, if used strategically, affect the welfare of other miners. As we saw, this generates adversarial incentives that lead to griefing and destabilize the Nash equilibrium outcome. However, as mining networks continue to grow a question that naturally arises is whether we can reason about miners' incentives when their individual influence becomes negligible on aggregate network levels. This requires a *market-theoretic* rather than game-theoretic perspective that will also consider mining in the presence of multiple blockchains. Further interesting questions involve the evaluation of griefing incentives under various practical scenarios (e.g., are larger miners more easy/difficult to grief or how much grief can a miner cause who is willing to incur a relative loss $x\%$ etc) or the design of alternative metrics that overcome the shortcomings of the griefing factors that were outlined in Remark 3.

A Omitted Proofs

Proof of Theorem 2 Part (i). For $\Delta < x^*$, Corollary 1 implies that

$$u_j(x^*) - u_j(x^* + \Delta, \mathbf{x}_{-i}^*) > u_i(x^*) - u_i(x_i^* + \Delta, \mathbf{x}_{-i}^*).$$

Since $u_i(x^*) = u_j(x^*)$ for all $i, j \in N$ by the symmetry assumption, $c_i = c > 0$ for all $i \in N$, it follows that $u_i(x_i^* + \Delta, \mathbf{x}_{-i}^*) - u_j(x^* + \Delta, \mathbf{x}_{-i}^*) > 0$ as claimed.

Part (ii). The own loss of miner i by deviating to allocation $x_i^* + \Delta$ when all other miners use their equilibrium allocations \mathbf{x}_{-i}^* is equal to

$$\begin{aligned} u_i(\mathbf{x}^*) - u_i(x_i^* + \Delta, \mathbf{x}_{-i}^*) &= \frac{x_i^*}{X^*} - c_i x_i^* - \left[\frac{x_i^* + \Delta}{X^* + \Delta} - c_i (x_i^* + \Delta) \right] \\ &= \Delta \left[c_i - \frac{X_{-i}^*}{X^* (X^* + \Delta)} \right]. \end{aligned}$$

By Theorem 1, $X^* = 1/c^*$, and $x_i^* = (1 - c_i/c^*)/c^*$, which implies that $X_{-i}^* = X^* - x_i^* = c_i/(c^*)^2$. Substituting in the right hand side of the above equality yields

$$u_i(\mathbf{x}^*) - u_i(x_i^* + \Delta, \mathbf{x}_{-i}^*) = \Delta \left[c_i - \frac{c_i/(c^*)^2}{(1/c^* + \Delta)/c^*} \right] = \frac{\Delta^2 c_i c^*}{1 + c^* \Delta}. \quad (8)$$

Similarly, the loss incurred to any miner $j \neq i$ by miner i 's deviation is equal to

$$\begin{aligned}
 u_j(\mathbf{x}^*) - u_j(x_i^* + \Delta, \mathbf{x}_{-i}^*) &= \frac{x_j^*}{X^*} - c_j x_j^* - \left[\frac{x_j^*}{X^* + \Delta} - c_j x_j^* \right] \\
 &= \frac{x_j^* \Delta}{X^* (X^* + \Delta)} \\
 &= \frac{1}{c^*} \left(1 - \frac{c_j}{c^*} \right) \cdot \frac{\Delta}{(1/c^* + \Delta)/c^*} \\
 &= \frac{\Delta (c^* - c_j)}{1 + c^* \Delta}. \tag{9}
 \end{aligned}$$

Since $c_j < c^*$ for all miners $j \in N$, the last expression is always positive (i.e., all miners incur a strictly positive loss). Summing over all $j \in N$ with $j \neq i$, Eq. (9) yields

$$\begin{aligned}
 \sum_{j \neq i}^n [u_j(\mathbf{x}^*) - u_j(x_i^* + \Delta, \mathbf{x}_{-i}^*)] &= \frac{\Delta}{1 + c^* \Delta} \left[(n-1)c^* - \sum_{j \neq i} c_j \right] \\
 &= \frac{\Delta}{1 + c^* \Delta} \left[(n-1)c^* - \sum_{j=1}^n c_j + c_i \right] \\
 &= \frac{\Delta c_i}{1 + c^* \Delta}, \tag{10}
 \end{aligned}$$

where the last equality holds by definition of c^* , cf. (3). Combining Eqs. (8) and (10), we obtain

$$\text{GF}(\mathbf{x}^*; (x_i^* + \Delta, \mathbf{x}_{-i}^*)) = \left(\frac{\Delta c_i}{1 + c^* \Delta} \right) / \left(\frac{\Delta^2 c_i c^*}{1 + c^* \Delta} \right) = \frac{1}{c^* \Delta},$$

which concludes the proof of part (ii).

Part (iii). For an allocation $\mathbf{y} = (y_i)_{i \in N}$ to be individually non-griefable it must hold that

$$u_j(\mathbf{y}) - u_j(y_i + \Delta, \mathbf{y}_{-i}) < u_i(\mathbf{y}) - u_i(y_i + \Delta, \mathbf{y}_{-i}),$$

for all $i, j \in N$ with $i \neq j$ and for all $\Delta > 0$. This yields the inequality (cf. Eq. (9) in the proof of part (ii))

$$\frac{y_j \Delta}{Y(Y + \Delta)} < \Delta \left[c_i - \frac{Y - y_i}{Y(Y + \Delta)} \right], \quad \text{for each } i, j \in N, \Delta > 0,$$

which after some trivial algebra can be equivalently written as

$$c_i(Y + \Delta)Y > Y + y_j - y_i, \quad \text{for each } i, j \in N, \Delta > 0.$$

Since the left hand side is increasing in Δ and since the above must hold for each $\Delta > 0$, it suffices to prove the inequality for $\Delta = 0$ in which case it must hold with equality. This gives the condition

$$c_i Y^2 = Y + y_j - y_i, \quad \text{for each } i, j \in N,$$

which can be now solved for the individually non-grievable allocation $\mathbf{y} = (y_i)_{i \in N}$. Summing over $j \neq i \in N$ yields

$$(n-1)c_i Y^2 = (n-1)Y + Y - y_i - (n-1)y_i, \quad \text{for each } i \in N,$$

or equivalently

$$y_i = Y \left[1 - \frac{n-1}{n} c_i Y \right], \quad \text{for each } i \in N. \quad (11)$$

Summing Eq. (11) over all i yields

$$Y = Y \left[n - \frac{n-1}{n} Y \sum_{i \in N} c_i \right]$$

which we can solve for Y to obtain that $Y = \frac{n}{\sum_{i \in N} c_i}$. Using the notation of Eq. (3), this can be written as

$$Y = \frac{n}{n-1} \cdot \frac{n-1}{\sum_{i \in N} c_i} = \frac{n}{n-1} \cdot \frac{1}{c^*}.$$

Substituting back in Eq. (11) yields the unique allocations y_i

$$\begin{aligned} y_i &= \frac{n}{(n-1)c^*} \left[1 - \frac{(n-1)c_i}{n} \frac{n}{(n-1)c^*} \right] \\ &= \frac{n}{n-1} (1 - c_i/c^*) / c^* = \frac{n}{n-1} x_i^*, \end{aligned}$$

where $x_i^* = (1 - c_i/c^*) / c^*$ is the Nash equilibrium allocation for each $i \in N$ (cf. Theorem 1). This concludes the proof of part (iii). \square

Proof of Proposition 1 Part (i). Let $\Delta_i > 0$ be such that $u_j(x_i^* + \Delta_i, \mathbf{x}_{-i}^*) = 0$. Then

$$\begin{aligned}
u_j(x_i^* + \Delta_1, \mathbf{x}_{-i}^*) = 0 &\implies \frac{x_j^*}{X^* + \Delta_1} - c_j x_j^* = 0 \\
&\implies \frac{1}{X^* + \Delta_1} = c_j \\
&\implies \Delta_1 = \frac{v}{c_j} - X^*
\end{aligned}$$

Since $X^* = \frac{1}{c^*}$ by Theorem 1, it follows that

$$\Delta_i = \frac{(c^* - c_i)}{(c_i c^*)} = \frac{1}{c_i} - \frac{1}{c^*}. \quad (12)$$

The previous equation implies in particular that $\Delta_i < \Delta_j$ if and only if $c_i > c_j$ for any $i \neq j \in N$.

Part (ii). From Eq. (10) in the proof of Theorem 2, we know that the absolute losses, $L(\Delta)$, of the network when miner i deviates to $x_i^* + \Delta$ are equal to

$$L(\Delta) = \sum_{j \neq i}^n u_j(x^*) - u_j(x_i^* + \Delta, \mathbf{x}_{-i}^*) = \frac{\Delta c_i}{1 + c^* \Delta}.$$

Taking the derivative of the right hand side expression with respect to Δ , we find that

$$\frac{\partial}{\partial \Delta} L(\Delta) = \frac{\partial}{\partial \Delta} \frac{\Delta c_i}{1 + c^* \Delta} = \frac{c_i}{(1 + c^* \Delta)^2} > 0.$$

This implies that the absolute losses of the network are increasing in Δ . Thus, for $\Delta \in (0, \Delta_i]$, they are maximized at $\Delta = \Delta_i$ where they are equal to

$$L(\Delta_i) = \frac{\left(\frac{1}{c_i} - \frac{1}{c^*}\right) c_i}{1 + c^* \left(\frac{1}{c_i} - \frac{1}{c^*}\right)} = c_i \cdot (1 - c_i/c^*)/c^* = c_i x_i^*,$$

where the last equality follows from Theorem 1. \square

Proof of Observation 1 Since $c_i < 1$ for all $i \in N$ (recall that this equivalent to $c_i < v$ prior to normalization which is naturally satisfied), it holds that $\sum_{i=1}^n c_i^2 < \sum_{i=1}^n c_i$. Along with the definition of c^* , cf. (3), this yields

$$\begin{aligned}
\sigma_c^2 &= \frac{1}{n-1} \sum_{i=1}^n (c_i - \bar{c})^2 = \frac{1}{n-1} \sum_{i=1}^n c_i^2 - \frac{1}{n(n-1)} \left(\sum_{i=1}^n c_i \right)^2 \\
&\leq \frac{1}{n-1} \sum_{i=1}^n c_i - \frac{n-1}{n} \left(\frac{1}{n-1} \sum_{i=1}^n c_i \right)^2 = c^* \left(1 - \frac{n-1}{n} c^* \right).
\end{aligned}$$

The participation constraint, $c_i < c^*$ for all $i \in N$, implies, in particular, that $c_{\max} < c^*$. Moreover, $c^* = \frac{1}{n-1} \sum_{i=1}^n c_i < \frac{n}{n-1} c_{\max}$. Substituting these in the last expression of the above inequality, we obtain that

$$\begin{aligned} \sigma_c^2 &< c^* \left(1 - \frac{n-1}{n} c^* \right) < \frac{n}{n-1} c_{\max} \left(1 - \frac{n-1}{n} c_{\max} \right) \\ &= c_{\max} \left(\frac{n}{n-1} - c_{\max} \right). \quad \square \end{aligned}$$

References

1. Alkalay-Houlihan, C., & Shah, N. (2019). The pure price of anarchy of pool block withholding attacks in bitcoin mining. *AAAI Conference on Artificial Intelligence, AAAI-19*, 33(1). <https://doi.org/10.1609/aaai.v33i01.33011724>.
2. Amegashie, J. (2012). Productive versus destructive efforts in contests. *European Journal of Political Economy*, 28(4), 461–468. <https://doi.org/10.1016/j.ejpoleco.2012.05.005>
3. Arnosti, N., & Weinberg, S.M. (2019). Bitcoin: A natural oligopoly. In A. Blum (ed.), *10th Innovations in Theoretical Computer Science Conference (ITCS 2019)*, 124(5), 1–5. <https://doi.org/10.4230/LIPIcs.ITCS.2019.5>.
4. Auer, R. (2019). Beyond the doomsday economics of proof-of-work in cryptocurrencies. Discussion Paper DP13506, London: Centre for Economic Policy Research. https://cepr.org/active/publications/discussion_papers/dp.php?dpno=13506.
5. Bentov, I., Gabizon, A., & Mizrahi, A. (2016). Cryptocurrencies without proof of work. In J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, & K. Rohloff (Eds.), *Financial Cryptography and Data Security* (pp. 142–157). Berlin Heidelberg, Berlin, Heidelberg: Springer.
6. Budish, E. (2018). The economic limits of bitcoin and the blockchain. Working Paper 24717, *National Bureau of Economic Research*. <https://doi.org/10.3386/w24717>.
7. Buterin, V. (2018). A grieving factor analysis model. <https://ethresear.ch/t/a-grieving-factor-analysis-model/2338ethresear.ch>. Accessed February 11, 2021.
8. Buterin, V., Reijbergen, D., Leonardos, S., & Piliouras, G. (2019). Incentives in ethereum’s hybrid casper protocol. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 236–244). USA: IEEE. <https://doi.org/10.1109/BLOC.2019.8751241>.
9. Chen, X., Papadimitriou, C., & Roughgarden, T. (2019). An axiomatic approach to block rewards. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (pp. 124–131). New York, NY, USA: AFT ’19, ACM. <https://doi.org/10.1145/3318041.3355470>.
10. Cheung, Y. K., Leonardos, S., & Piliouras, G. (2021). Learning in markets: Greed leads to chaos but following the price is right. In Z. H. Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence* (pp. 111–117). IJCAI-21. International Joint Conferences on Artificial Intelligence Organization, virtual. <https://doi.org/10.24963/ijcai.2021/16>.
11. Dimitri, N. (2017). *Bitcoin mining as a contest*. *Ledger*, 2, 31–37. <https://doi.org/10.5195/ledger.2017.96>
12. DiPalantino, D., & Vojnovic, M. (2009). Crowdsourcing and all-pay auctions. In (pp. 119–128). EC ’09. <https://doi.org/10.1145/1566374.1566392>.
13. Eyal, I., & Sirer, E. G. (2018). Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM*, 61(7), 95–102. <https://doi.org/10.1145/3212998>

14. Ferreira, M. V. X., Moroz, D.J., Parkes, D. C., & Stern, M. (2021). Dynamic posted-price mechanisms for the blockchain transaction-fee market. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies* (pp. 86–99). New York, NY, USA: AFT '21. Association for Computing Machinery. <https://doi.org/10.1145/3479722.3480991>.
15. Fiat, A., Karlin, A., Koutsoupias, E., & Papadimitriou, C. (2019). Energy equilibria in proof-of-work mining. In (pp. 489–502). New York, NY, USA: EC'19. ACM. <https://doi.org/10.1145/3328526.3329630>.
16. Gandal, N., & Gans, J. (2019). More (or less) economic limits of the blockchain. Discussion Paper DP14154. London: Centre for Economic Policy Research. https://cepr.org/active/publications/discussion_papers/dp.php?dpno=14154.
17. Garay, J., Kiayias, A., & Leonardos, N. (2015). The bitcoin backbone protocol: Analysis and applications. In E. Oswald, & M. Fischlin (Eds.), *Advances in Cryptology - EUROCRYPT* (pp. 281–310). Berlin: Springer https://doi.org/10.1007/978-3-662-46803-6_10.
18. Gersbach, H., Mamageishvili, A., & Schneider, M. (2020). Vote delegation and malicious parties. In *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 1–2). <https://doi.org/10.1109/ICBC48266.2020.9169391>
19. Gersbach, H., Mamageishvili, A., & Schneider, M. (2022). Staking pools on blockchains. <https://doi.org/10.48550/ARXIV.2203.05838>.
20. Hehenkamp, B., Leininger, W., & Possajennikov, A. (2004). Evolutionary equilibrium in Tullock contests: spite and overdispersion. *European Journal of Political Economy*, 20(4), 1045–1057. <https://doi.org/10.1016/j.ejpolco.2003.09.002>
21. Horton, J. J., & Chilton, L. B. (2010). The labor economics of paid crowdsourcing. In *Proceedings of the 11th ACM Conference on Electronic Commerce* (pp. 209–218). EC '10. <https://doi.org/10.1145/1807342.1807376>
22. Kiayias, A., Koutsoupias, E., Kyropoulou, M., & Tselekounis, Y. (2016). Blockchain mining games. In *Proceedings of the 2016 ACM Conference on Economics and Computation* (pp. 365–382). New York, NY, USA: EC '16, ACM. <https://doi.org/10.1145/2940716.2940773>.
23. Koki, C., Leonardos, S., & Piliouras, G. (2020). Do cryptocurrency prices camouflage latent economic effects? A bayesian hidden markov approach. *Future Internet*, 28(1), 5. <https://doi.org/10.3390/fi12030059>
24. Koki, C., Leonardos, S., & Piliouras, G. (2022). Exploring the predictability of cryptocurrencies via Bayesian hidden Markov models. *Research in International Business and Finance*, 59, 101554. <https://doi.org/10.1016/j.ribaf.2021.101554>
25. Konrad, K. A. (2000). Sabotage in rent-seeking contests. *Journal of Law, Economics, & Organization*, 16(1), 155–165. <https://doi.org/10.2307/3555012>
26. Kwon, Y., Liu, J., Kim, M., Song, D., & Kim, Y. (2019). Impossibility of full decentralization in permissionless blockchains. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies* (pp. 110–123). New York, NY, USA: AFT '19, Association for Computing Machinery. <https://doi.org/10.1145/3318041.3355463>.
27. Leonardos, N., Leonardos, S., & Piliouras, G. (2020). Oceanic games: Centralization risks and incentives in blockchain mining. In P. Pardalos, I. Kotsireas, Y. Guo, & W. Knottenbelt (Eds.), *Mathematical Research for Blockchain Economy* (pp. 183–199). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-37110-4_13.
28. Leonardos, S., & Melolidakis, C. (2020). Endogenizing the cost parameter in cournot oligopoly. *International Game Theory Review*, 22(02), 2040004. <https://doi.org/10.1142/S0219198920400046>
29. Leonardos, S., Monnot, B., Reijsbergen, D., Skoulakis, S., & Piliouras, G. (2021). Dynamical analysis of the EIP-1559 ethereum fee market. In *Proceedings of the 3rd ACM conference on Advances in Financial Technologies*. New York, NY, USA: AFT '21, Association for Computing Machinery.
30. Leonardos, S., Reijsbergen, D., & Piliouras, G. (2019). Weighted voting on the blockchain: Improving consensus in proof of stake protocols. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 376–384). <https://doi.org/10.1109/BLOC.2019.8751290>.

31. Leonardos, S., Reijnsbergen, D., & Piliouras, G. (2020). PREStO: A systematic framework for blockchain consensus protocols. *IEEE Transactions on Engineering Management*, 67(4), 1028–1044. <https://doi.org/10.1109/TEM.2020.2981286>
32. Levin, D., LaCurts, K., Spring, N., & Bhattacharjee, B. (2008). Bittorrent is an auction: Analyzing and improving bittorrent’s incentives. *SIGCOMM Computer Communication Review*, 38(4), 243–254. <https://doi.org/10.1145/1402946.1402987>
33. Shen, M. (2020). Crypto investors have ignored three straight 51% attacks on ETC. <https://www.coindesk.com/crypto-51-attacks-etc-coindesk.com>. Accessed February 11, 2021.
34. Monnot, B., Hum, Q., Koh, C. M., & Piliouras, G. (2020). Ethereum’s transaction fee market reform in eip 1559. WINE 20, Workshop on Game Theory in Blockchain. https://econcs.pku.edu.cn/wine2020/wine2020/Workshop/GTiB20_paper_7.pdf.
35. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf>. Accessed August 31, 2020.
36. Reijnsbergen, D., Sridhar, S., Monnot, B., Leonardos, S., Skoulakis, S., & Piliouras, G. (2021). Transaction fees on a honeymoon: Ethereum’s eip-1559 one month later. In *2021 IEEE International Conference on Blockchain (Blockchain)* (pp. 196–204). <https://doi.org/10.1109/Blockchain53845.2021.00034>.
37. Schaffer, M. E. (1988). Evolutionarily stable strategies for a finite population and a variable contest size. *Journal of Theoretical Biology*, 132(4), 469–478. [https://doi.org/10.1016/S0022-5193\(88\)80085-7](https://doi.org/10.1016/S0022-5193(88)80085-7)
38. Singh, R., Dwivedi, A. D., Srivastava, G., Wiszniewska-Matyszek, A., & Cheng, X. (2020). A game theoretic analysis of resource mining in blockchain. *Cluster Computing*, 23(3), 2035–2046. <https://doi.org/10.1007/s10586-020-03046-w>
39. Stoll, C., Klaaßen, L., & Gallsdörfer, U. (2019). *The carbon footprint of bitcoin*. *Joule*, 3(7), 1647–1661. <https://doi.org/10.1016/j.joule.2019.05.012>
40. Sun, J., Tang, P., & Zeng, Y. (2020). Games of miners. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems* (pp. 1323–1331). Richland, SC: AAMAS ’20, International Foundation for Autonomous Agents and Multiagent Systems. <https://doi.org/10.5555/3398761.3398914>.
41. Wave Financial LLC. (2021). Ethereum 2.0 staking, a worthwhile investment? <https://www.cityam.com/ethereum-2-0-staking-a-worthwhile-investment/cityam.com>. Accessed February 11, 2021.
42. Wikipedia Contributors. (2021). Griefer—wikipedia, the free encyclopedia. <https://en.wikipedia.org/w/index.php?title=Griefer&oldid=1006081077>. Accessed February 11, 2021.

Data-Driven Analysis of Central Bank Digital Currency (CBDC) Projects Drivers



Toshiko Matsui and Daniel Perez

Abstract In this paper, we use a variety of machine learning methods to quantify the extent to which economic and technological factors are predictive of the progression of Central Bank Digital Currencies (CBDC) within a country, using as our measure of this progression the CBDC project index (CBDCPI). By extracting and aggregating cross country data provided by several international organisations, we find that the financial development index is the most important feature for our model, followed by the GDP per capita and an index of the voice and accountability of the country's population. Our results are consistent with previous qualitative research which finds that countries with a high degree of financial development or digital infrastructure have more developed CBDC projects. Further, we obtain robust results when predicting the CBDCPI at different points in time.

Keywords Central bank digital currency (CBDC) · Digital currency · CBDC project index · Machine learning · Multilayer perceptron · Random forest

1 Introduction

Recent advances in financial technology and distributed ledgers [17, 21] have paved the way to the extensive use of digital currencies. Although the advancement of these currencies came from private initiatives such as Bitcoin [20], Ethereum [28], and Libra [11, 13], researchers and policymakers are contemplating whether central banks can also issue their own digital currencies, usually referred to as *central bank digital currency* (CBDC).

There has been a great deal of discussion about the implications of introducing CBDCs. Although the concept of a CBDC has existed for quite a long time [27], the

T. Matsui (✉) · D. Perez

Department of Computing, Imperial College London, London, United Kingdom
e-mail: t.matsui19@imperial.ac.uk

D. Perez

e-mail: daniel.perez@imperial.ac.uk

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_6

95

stance toward whether central banks should introduce them has changed drastically over the past year. Initially the focus of central banks was on systemic implications [7] but several factors deriving from the benefits of the digital money have recently motivated central banks to issue CBDCs. This trend has further been fueled by the declining use of cash due to the growth of cashless payments, the possible introduction of global stablecoins [16] and the Covid-19 pandemic [3].

In fact, a great number of central banks are undertaking extensive work on CBDC [9], several of which have issued research or statements on the related motivations, architectures, risks, and benefits. For instance, Boar et al. [9] refer to the observed shift to intensive practical development from conceptual research found in emerging markets, driven by stronger motivations than those of advanced economy central banks. In practice, several central banks issued their CBDCs between August and December 2020 (see Sect. 2.2). Further, a few central banks are aiming to issue their CBDCs in the next few years, which is attracting a lot of attention [4]; for example, the beginning of 2021 saw the closing of the ECB digital euro consultation with record level of public feedback [14]. This move is consistent with the observation that the introduction of CBDCs can present significant innovations in money and banking history [15]. Central banks are continuously pondering whether to issue their own digital currencies to the public – in this context, recent papers [2, 24] further examine the countries at the frontier of issuing CBDCs in order to share insights, lessons, and remaining issues, to help the many countries following in their footsteps.

However, despite the great amount of analysis conducted regarding the important questions surrounding CBDCs, relatively little quantitative analysis has been undertaken especially on the drivers of the CBDC projects. The previous research include the potential risk and benefits of introducing CBDC, and quantitatively, the welfare gains of bringing CBDC into the economy [12]. Of the relevant research that exists in this vein, [4] suggests, by taking an ordered probit approach [19] with a comprehensive cross country database, that the majority of the CBDC projects are found in digitised economies with a high capacity for innovation. They conclude that some of the potential drivers of CBDC development are related to factors affecting a country's digital infrastructure, innovation capacity, institutional quality, development and financial inclusion, public interest in CBDCs, and cross-border transactions.

This study examines the economic and institutional drivers of CBDC projects by applying machine learning techniques to the related variables obtained from official sources that are available for a wide cross section of countries. Our primary objective is to improve the understanding of the dominant drivers for CBDCs and the factors that increase the possibly of a country to accelerate this effort. We use the CBDC project index (CBDCPI) [4] as our objective variable and factors affecting a country's digital or technological capability and government effectiveness as independent variables, in order to reduce the problem to identifying the independent variables with the most predictive power. To accomplish this, we utilise machine learning techniques to predict the CBDCPI and pick the most important variables for our model.

We compare two types of classifiers that are able to learn non-linear functions: a multilayer perceptron (MLP) [22, 23] and a random forest [10]. In the experiment, we find random forest performs better than MLP, and that the financial development

index [25] is the most important feature for our model, followed by the GDP per capita and the voice and accountability, when explaining the CBDCPI drivers for August 2020. This concurs with [4], which concluded that more developed CBDC projects can be found in countries with higher financial development index, digital infrastructure, GDP, and institutional characteristics. As a robustness check, we have performed the same analysis with full and aggregated data and with December 2020 CBDCPI. Results are broadly consistent, although there were some minor changes in the ranks of important features.

This paper is structured as follows. Section 2 describes the preliminaries including the overview of CBDCs and the CBDC project index (CBDCPI). We subsequently present the empirical models and data in Sect. 3, then results in Sect. 4. We conclude in Sect. 5.

2 Preliminaries

In this section, we provide an overview of the key concepts from central bank digital currency (CBDC) relevant to this paper.

2.1 Central Bank Digital Currency (CBDC)

A central bank digital currency (CBDC) is the digital form of the fiat currency of a particular nation (or region). It differs from virtual currency and cryptocurrency, as CBDC is issued by the state and possesses the legal tender status declared by the government [18]. Examples include the Digital Currency/Electronic Payments (DC/EP) by China's central bank and e-krona by the central bank of Sweden.

The introduction of CBDCs is receiving more attention than ever before. Although the concept of a CBDC was already proposed decades ago [27], recent IT progress and its application to the financial industry have motivated central banks and academics to study the risks and benefits of making CBDC accessible to the general public [6, 8], as presented in Table 1 [1].

Table 1 Benefits and challenges of CBDCs

Advantages	Disadvantages
Low cost of cash	Higher run risk
Enhances financial inclusion	Commercial bank disintermediation
Stabilises the payment system	Enhances currency substitution
Faster monetary policy transmission	Risk and cost for central banks

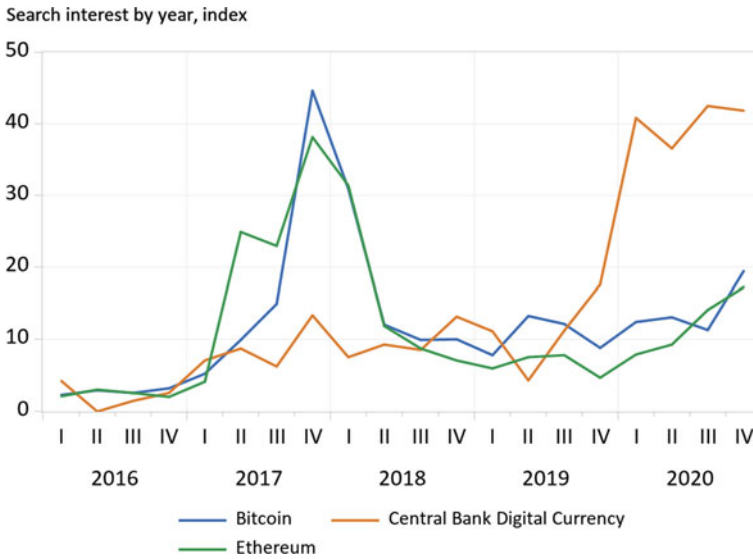


Fig. 1 Google Trends over time

Further, attitudes towards whether CBDCs should be issued by central banks have changed drastically over the past year. This derived from the diminishing use of cash due to the rise of cashless payments, the possible entry of global stablecoins [16] and the Covid-19 pandemic. Specifically, social distancing measures and public concerns that cash may transmit the Covid-19 virus and novel government-to-person payment schemes have further fueled the shift toward digital payments, and may act as a driver of CBDC projects [3].

In fact, CBDCs have gained global attention, not only within central bank communities but also by the public. Figure 1 charts the Google search interest over time.¹ This shows that the current interest in CBDC is increasing, reaching a level almost as high as Bitcoin’s during its price spike of 2017.

However, a majority of CBDCs are still in research or pilot stage, although a survey in early 2020 showed more than 80% of central banks were studying the subject [26]. Further, the drivers of the CBDC projects are yet to be thoroughly investigated [4].

2.2 CBDC Project Index (CBDCPI)

The *CBDC project index* (CBDCPI) was firstly proposed by Auer et al. [4] to measure the central bank’s progress toward the development of a retail or wholesale CBDC.

¹ We took 12-week moving average of Google Trends search results.

CBDCPI represents publicly announced work by central banks on CBDC related projects. The index takes a value between 0 and 4 defined as follows:

- 0 - No announced project
- 1 - Public research studies
- 2 - Ongoing or completed pilot
- 3 - Live CBDC

There are two sub-indices, one for retail and one for wholesale CBDC projects. Wholesale CBDC is devised as a new instrument for settlement between financial organisations, whereas retail CBDC aims to replace cash with the properties of central bank liability. The overall index for a country is the maximum of these two sub-indices.

According to the dataset provided by the online annex of [4], there was no country with index 3 (live CBDC) as of August 2020 but by the end of 2020, seven countries and jurisdictions, including Bahamas, Canada, Switzerland, France, have shifted to 3.² This clearly reflects the recent trend and discussion that have brought the analysis about digital money and CBDC to the fore [5].

3 Data and Methodology

Our main goal is to understand better the main drivers for CBDCs and the factors that makes a country more or less likely to push this effort. Using the CBDC project index (CBDCPI) as our objective variable and factors affecting a country's digital or technological capability and government effectiveness as explanatory variables, we can reduce this to finding the explanatory variables with the most predictive power. To achieve this, we leverage machine learning techniques to predict the CBDCPI and extract the most important variables for our model. We use the rest of this section to describe our data and provide details about our machine-learning based methodology.

3.1 Data

We extract our dataset from the World Bank, the International Monetary Fund (IMF), and the data source of a BIS working paper [4]. After having processed and cleaned the data, it ended up containing 16 variables, including data from 2000 to 2019, with more than 170 countries and jurisdictions. The Table 2 lists the observed variables included in our analysis.

Our variables can be divided into the following categories: (i) digital infrastructure, (ii) development and financial inclusion, (iii) institutional characteristics,

² The updated CBDC projects status is available in an online annex of [4] (See <https://www.bis.org/publ/work880.htm>). The information is said to have been collected through desk research and with the help of contacts at several individual central banks.

Table 2 Table of observed variables

Variable	Description	Source
Digital infrastructure		
Mobile subscriptions	Subscriptions to a mobile telephone service (per 100 ppl.)	WB
Secure Internet	The number of secure Internet servers (per 1 million ppl.)	WB
Broadband subscription	Fixed broadband subscriptions (per 100 ppl.)	WB
indiv. Internet use	Individuals using the Internet (% of population)	WB
Development and financial inclusion		
Account ownership	Account ownership at a financial institution or with a mobile-money-service provider (% of population aged 15+)	WB
FD index	Indices that summarise the degree of developments of financial institutions and financial markets	IMF
GDP per capita	GDP divided by midyear population (USD)	WB
Institutional characteristics		
Government effectiveness	Quality of public services, policy implementation etc.	WB
Regulatory quality	Ability to formulate and implement sound policies etc.	WB
Voice and accountability	Extent of citizens' participation and freedom expression	WB
Innovation environment		
Access to electricity	Access to electricity (% of population)	WB
Demographic characteristics		
% of people over 65	Total population 65 years of age or older	WB
Cross-border transactions		
Trade (% of GDP)	Sum of exports and imports (% of GDP)	WB

Source World Bank (DataBank), IMF (working paper [25])

Dependent variable, CBDCPI, was obtained from BIS (online annex of [4])

Table 3 Countries with a CBDCPI of 3 as of December 2020

Country	Overall (Dec 20)	Overall (Aug 20)	Retail (Dec 20)	Wholesale (Dec 20)
Bahamas	3	2	3	0
Canada	3	2	1	2
Switzerland	3	1	1	2
Euro area (ECB)	3	2	1	2
France	3	2	1	2
Japan	3	2	1	2
South Africa	3	1	1	2

Source Online annex of [4]

(iv) innovation environment, (v) demographic characteristics, and (vi) cross-border transactions. Each variable contains several years of data ranging from 2000 to 2019. Although our dataset on financial development index holds data about several financial development sub-indices in terms of their depth, access, and efficiency, we only include the top-level index.³ We perform our analysis both using the *full data* and an aggregated version of our data. For the *aggregated data*, we average each variable over the period 2014-2019, subject to data availability.

The CBDCPI has 176 observations, one per country, each taking the value of 0, 1, 2, or 3, as described in Sect. 2.2. We obtain the CDCPI for December 2020, in addition to its August 2020 value from previous research [4]. The major difference between these two variables is that, as of August 2020, there were no countries with a live CBDC (i.e. no index with value 3) while there were 7 of them with a live CBDC by December 2020. We show the countries with a CDCPI of 3 as of December 2020 in Table 3.

3.2 Methodology

Instead of the ordered probit approach [19] applied in [4], we model the problem as a classification task where the goal is to predict the CBDCPI given the set of input variables described above. Given that the CBDCPI is a value between 0 and 3, it is easy to model as a categorical variable.

We settle on a random forest as our primary model, as it is known to be able to learn complex non-linear functions while being interpretable enough to extract the most important input variables [10]. To obtain a point of comparison for the predictions of our random forest, we also train a multilayer perceptron [22, 23] on the same task. However, given that multilayer perceptrons are not interpretable enough to understand the most important features, we only use these results for comparison.

³ See Financial Development Index Database by IMF for more information. (<https://data.imf/?sk=f8032e80-b36c-43b1-ac26-493c5b1cd33b>).

We utilise this methodology with the full version and the aggregated version of the data to predict the CBDCPI both in August and December 2020.

4 Results

This section presents the results we obtained by training the models described above on our dataset.

Before starting our training process, we preprocess the data to filter out lacking data. We first remove all the countries for which the CBDCPI is not available, as well as the countries for which one or more of the observed variables is not available (e.g. do not have a single year of data). When a country is missing a year for a particular variable, we use the previous year to fill for it (e.g. if the GDP per capita is available for 2018 but not 2019, we set it to the value of 2019). After this filtering process, we obtain a final list of 145 countries with only 6 countries having a CBDCPI of 3 as of December 2020; unfortunately, the Euro Area was not included in the final dataset as we did not have the financial development index data for it. Further, we obtain a total of 13 variables for our aggregated data and 135 variables for the full data.

Then, we randomly split our dataset in two equal splits for training and testing. We then tune the hyper-parameters of our two models. We note that given the small size of our dataset, we use the full data instead of having a separate cross-validation set. We find that our random forest works best with a total of 100 estimators. For our multilayer perceptron, we use two layers, the first one with a number of neurons equal to the number of features and the second one a fixed size of 10 neurons. Finally, we train the two models on our training data and evaluate them on our test data. We present the accuracy of the two models in Table 4. Overall, our model performs better with the full data rather than its aggregated version.

Next, we use the features extracted by our random forest to understand better what the potential drivers of CBDC are. We use the aggregated August 2020 data to compare with [4] and see if the random forest achieves similar results as to the drivers for CBDCs. We find that the financial development index [25] is by far the

Table 4 Accuracy of different classifiers on full and aggregated data

	Classifier	Full data		Aggregated data	
		Train	Test	Train	Test
Aug	MLP	1.0	0.79	0.99	0.74
	Random Forest	1.0	0.78	1.0	0.77
Dec	MLP	1.0	0.67	1.0	0.62
	Random Forest	1.0	0.78	1.0	0.68

Table 5 Most important features for the random forest (Aug 2020, aggregated)

Feature	Importance
Financial development index	0.165
GDP per capita	0.098
Voice and accountability	0.095
Broadband subscriptions (per 100 people)	0.093
Government effectiveness	0.087

Table 6 Most important features for the random forest (Dec 2020, aggregated)

Feature	Importance
% of people over 65	0.134
Financial development index	0.109
Mobile cellular subscriptions (per 100 people)	0.101
GDP per capita	0.090
Voice and accountability	0.089

most important feature for our model, followed by the GDP per capita and the voice and accountability, when explaining the CBDCPI drivers for August 2020. This is consistent with [4], stating that the CBDC projects to be more developed where there is higher financial development index, GDP, digital infrastructure, and institutional characteristics such as government effectiveness and voice and accountability. We summarize the top 5 features and their importance for random forest in Table 5.

For robustness check, we conduct the same analysis with December 2020 CBDCPI data as an objective variable. The results are almost consistent with the August CBDCPI data, aside from the fact that the aging rate additionally accounts for the December 2020 data – random forest performing much better and allows to extract the most important features used for classification. The most significant features are: mean 65+, financial development index, mobile cellular subscription, GDP per capita, and voice and accountability, showing that the main features predicted as important are proved to be important with December 2020 CBDCPI data, as suggested in Table 6.

5 Conclusion

In this paper, by applying a variety of machine learning methods used to learn complex non-linear functions for comprehensive cross country data, we investigated the importance of each economic and technological factors in predicting the progression of Central Bank Digital Currencies (CBDC) project within a country, using as our measure of this advancement the CBDC project index (CBDCPI). We found that a

financial development index is the most important feature for our model, followed by the GDP per capita and an index of the voice and accountability of the country's population. Additionally, we confirmed that our results are in accordance with previous qualitative research which finds that countries with a high degree of financial development or digital infrastructure have more advanced CBDC projects. Moreover, we achieved robust results when examining the CBDCPI at different points in time.

A Appendix

This annex gives additional tables, regression results and figures to complement the paper. See main text for further discussion.

A.1 CBDC Projects Status

Below shows the part of the updated project score of global CBDC development efforts, relating to [4] (as of December 2020).⁴ Note that only the countries with index of 3 (live CBDC) and 2 (pilot) as of December 2020 are listed here.

Country	Overall*	Overall (Aug 20)	Retail*	Wholesale*
Bahamas	3	2	3	0
Canada	3	2	1	2
Switzerland	3	1	1	2
Euro area (ECB)	3	2	1	2
France	3	2	1	2
Japan	3	2	1	2
South Africa	3	1	1	2
United Arab Emirates	2	2	0	2
Australia	2	1	1	1
China	2	2	2	0
Ecuador	2	2	2	0
Eastern Caribbean	2	2	2	0
United Kingdom	2	2	1	1
Hong Kong	2	2	0	2
Indonesia	2	1	1	1
India	2	0	1	1
South Korea	2	2	2	0
Saudi Arabia	2	2	0	2
Sweden	2	2	2	0
Singapore	2	2	0	2
Swaziland	2	1	1	1
Thailand	2	2	0	2
Ukraine	2	2	2	0
Uruguay	2	2	2	0

*As of December 2020.

⁴ The dataset includes all projects announced as of 1 December 2020. For more information, see <https://www.bis.org/publ/work880.htm>.

Table 7 Most important features for the random forest classifier (Aug 2020)

Feature	Importance
Financial Development Index	0.165
GDP per capita	0.098
Voice and accountability	0.095
Broadband subscriptions (per 100 people)	0.093
Government effectiveness	0.087
% of people over 65	0.082
Individuals using the Internet (% of population)	0.080
Trade (% of GDP)	0.078
Secure Internet servers (per 1 million people)	0.072
Regulatory quality	0.068

A.2 Top 10 Features for the Random Forest Classifier with Aggregated Data

Tables 7 and 8 give the 10 most important independent variables for the random forest classifier with aggregated data (data averaged over the period 2014–19, subject to data availability), with August 2020 and December 2020 CBDCPI data as an objective variable, respectively.

Table 8 Most important features for the random forest classifier (Dec 2020)

Feature	Importance
% of people over 65	0.134
Financial Development Index	0.109
Mobile cellular subscriptions (per 100 people)	0.101
GDP per capita	0.090
Voice and accountability	0.089
Secure Internet servers (per 1 million people)	0.086
Individuals using the Internet (% of population)	0.080
Government effectiveness	0.080
Broadband subscriptions (per 100 people)	0.076
Trade (% of GDP)	0.067

Table 9 Most important features for the random forest classifier (Aug 2020)

Feature	Importance
Financial Development Index	0.052
Government effectiveness [YR2019]	0.033
Government effectiveness [YR2018]	0.020
Broadband subscriptions (per 100 people) [YR2017]	0.020
Individuals using the Internet (% of population) [YR2012]	0.020
Individuals using the Internet (% of population) [YR2015]	0.020
GDP per capita [YR2016]	0.018
Government effectiveness [YR2015]	0.016
Mobile cellular subscriptions (per 100 people) [YR2016]	0.015
% of people over 65 [YR1990]	0.014

A.3 Top 10 Features for the Random Forest Classifier with Full Data

Tables 9 and 10 show the 10 most important index for the random forest classifier with full data, with August 2020 and December 2020 CBDCPI data as an objective variable, respectively.

Table 10 Most important features for the random forest classifier (Dec 2020)

Feature	Importance
Financial Development Index	0.037
% of people over 65 [YR1990]	0.035
Mobile cellular subscriptions (per 100 people) [YR2019]	0.025
% of people over 65 [YR2018]	0.023
% of people over 65 [YR2017]	0.021
% of people over 65 [YR2000]	0.020
Government effectiveness [YR2017]	0.020
% of people over 65 [YR2015]	0.020
% of people over 65 [YR2013]	0.019
Mobile cellular subscriptions (per 100 people) [YR2015]	0.018

References

1. Adrian, T., & Mancini-Griffoli, T. (2019). Central bank digital currencies: 4 questions and answers, IMF Blog.
2. Agur, I., Anil, A., & Dell'Ariccia, G. (2022). Designing central bank digital currencies. *Journal of Monetary Economics*, forthcoming.
3. Auer, R., Cornelli, G., & Frost, J. (2020). Covid-19, cash and the future of payments. *BIS Bulletin*, 3.
4. Auer, R., Cornelli, G., & Frost, J. (2020). Rise of the central bank digital currencies: drivers, approaches and technologies. BIS working paper, No. 880.
5. Bank for International Settlement. (2020). Central bank group to assess potential cases for central bank digital currencies. BIS press release, 21 Jan 2020. <https://www.bis.org/press/p200121.htm>.
6. Bank for International Settlement. (2018). *Central bank digital currencies* (p. 174). Markets committee papers, No: CPMI.
7. Barontini, C., & Holden, H. (2019). *Proceeding with caution-a survey on central bank digital currency* (p. 101). No: BIS papers.
8. Bindseil, U. (2020). Tiered CBDC and the financial system. Working paper series 2351, European Central Bank. <https://ideas.repec.org/p/ecb/ecbwps/20202351.html>.
9. Boar, C., Holden, H., & Wadsworth, A. (2020). *Impending arrival-a sequel to the survey on central bank digital currency* (p. 107). No: BIS papers.
10. Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5–32. <https://link.springer.com/article/10.1023/A:1010933404324>.
11. Brühl, V. (2019). Libra—a differentiated view on facebook’s virtual currency project. CFS working paper series 633, Frankfurt a. M. <http://hdl.handle.net/10419/206412>
12. Davoodalhosseini, S. M. R. (2018). Central bank digital currency and monetary policy. Staff working papers, Bank of Canada. <https://EconPapers.repec.org/RePEc:bca:bocawp:18-36>.
13. Diem Association. (2020). Diem white paper. <https://www.diem.com/en-us/white-paper>.
14. European Central Bank. (2021). ECB digital euro consultation ends with record level of public feedback. ECB press release, 13 Jan 2021. <https://www.ecb.europa.eu/press/pr/date/2021/html/ecb.pr210113~ec9929f446.en.html>.
15. Fernández-Villaverde, J., Sanches, D., Schilling, L., & Uhlig, H. (2020). Central bank digital currency: Central banking for all? *Review of Economic Dynamics*. <http://www.sciencedirect.com/science/article/pii/S1094202520301150>.
16. Financial Stability Board. (2020). Regulation, supervision and oversight of “global stablecoin” arrangements. *Financial Stability Board*.
17. von zur Gathen, J. (2015). *CryptoSchool*. Berlin: Springer-Verlag. <https://link.springer.com/book/10.1007/978-3-662-48425-8>.
18. Grym, A., Heikkinen, P., Kauko, K., & Takala, K. (2017). *Central bank digital currency*. Banque de France: Tech. rep.
19. McKelvey, R.D., & Zavoina, W. (1975). A statistical model for the analysis of ordinal level dependent variables. *The Journal of Mathematical Sociology*, 4(1), 103–120. <https://www.tandfonline.com/doi/abs/10.1080/0022250X.1975.9989847>.
20. Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system. <http://www.bitcoin.org/bitcoin.pdf>.
21. Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2016). *Bitcoin and cryptocurrency technologies: A comprehensive introduction*. USA: Princeton University Press.
22. Rosenblatt, F. (1961). *Principles of neurodynamics: Perceptions and the theory of brain mechanism*. Spartan Books.
23. Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning internal representations by error propagation, pp. 318–362. Cambridge, MA: MIT Press.
24. Soderberg, G., et al. (2022). Behind the scenes of central bank digital currency emerging trends, insights, and policy lessons. FinTech Notes No. 2022/004, IMF. <https://www.imf.org/en/Publications/fintech-notes/Issues/2022/02/07/Behind-the-Scenes-of-Central-Bank-Digital-Currency-512174>.

25. Svirydzenka, K. (2016). Introducing a new broad-based index of financial development. IMF working papers, No. 15.
26. Economist, The. (2020). *Will central-bank digital currencies break the banking system?* (pp. 0013–0613). ISSN: Tech. rep. Dec.
27. Tobin, J. (1987) The case for preserving regulatory distinctions. *Proceedings of the Economic Policy Symposium*, 167–83.
28. Wood, G., et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014), 1–32.

Dissimilar Redundancy in DeFi



Daniel Perez and Lewis Gudgeon

Abstract The meteoric rise of Decentralized Finance (DeFi) has been accompanied by a number of financially devastating attacks on its protocols. There have been over 70 exploits of DeFi protocols, with the total of lost funds amounting to approximately 1.5bn USD. In this paper, we introduce a new approach to minimizing the frequency and severity of such attacks: dissimilar redundancy for smart contracts. In a nutshell, the idea is to implement a program logic *more than once*, ideally using *different* programming languages. Then, for each implementation, the results should *match* before allowing the state of the blockchain to change. This is inspired by and has clear parallels to the field of avionics, where on account of the safety-critical environment, flight control systems typically feature multiple redundant implementations. We argue that the high financial stakes in DeFi protocols merit a conceptually similar approach, and we provide a novel algorithm for implementing dissimilar redundancy for smart contracts.

1 Introduction

Decentralized Finance (DeFi) Protocol hacks are frequent—with more than 70 to date totalling losses of more than 1.5bn USD [8]. For a financial infrastructure that is purportedly going to replace traditional finance, this is worrisome. The severity of the issue of hacks is exacerbated by the non-custodial nature of DeFi systems. Unlike in traditional financial systems, where in the event of financial disaster, there are often safety-nets such as the state or insurers, in the DeFi setting there are no such safety provisions at scale. Moreover, while there is a nascent insurance market for DeFi insurance, e.g. [15], such solutions provide only a (necessary) second-best solution: providing coverage in the event of a DeFi system failure. A first-best solution is to prevent the failure in the first place.

D. Perez (✉) · L. Gudgeon
Imperial College London, London, UK
e-mail: daniel.perez@imperial.ac.uk

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes
in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_7

109

Preventing such failures is challenging. Leaving aside the security of the underlying blockchain layer, the two main pillars of DeFi protocol security are (i) extensive smart contract testing and (ii) code audits. Both of these have drawbacks. Ensuring adequate test coverage is very challenging, not least when the objective is to ensure all edge-cases are covered in the tests. While testing is an essential part of smart contract development, we suggest that it is not realistic to expect even best practice testing with a very high degree of coverage to be sufficient to prevent *all* bugs. Code audits are often performed by external teams under time pressure, with audits typically lasting 2–3 weeks from start to finish, even for teams with no prior familiarity with the code base. Even for the most experienced software engineer, native to DeFi, it is wishful thinking to believe that they will always be able to catch every bug. The problem is compounded by smart contract composability—where DeFi protocols are snapped together like DeFi lego—which serves to increase the challenge as tests and auditors now have to anticipate bugs that could arise with as yet unseen smart contracts.

This paper presents a new approach to preventing failures at the smart contract level: dissimilar redundancy. In aviation, the safety-critical nature has led to the emergence of a practice of implementing multiple separate and redundant flight control systems. For example, the Boeing 777 [24] featured a Fly-By-Wire flight control system that had to meet extremely high levels of functional integrity and reliability. To do this, it had three separate primary flight computers, with each computer containing three *dissimilar* internal computational lanes. The lanes differed in terms of compilers, power supply units and microprocessors, with, for example, lane 1 using the AMD 29050, lane 2 the Motorola 68040 and lane 3 the Intel 80486. Within each of the three flight computers, two of the lanes acted as monitors while the third lane was in command. In this way, the flight computer features a form of redundancy that is *dissimilar*, with the multiple lanes being resistant to bugs induced by microprocessors or compilers.

We apply the core of this idea to smart contracts. We implement and detail a system based on a proxy pattern which relies on dissimilar implementations of a DeFi protocol, and cross-checks one against the other before effecting any on-chain state change. On Ethereum, this approach has already been taken for client implementations, with the community maintaining multiple open-source clients, developed by different teams and using different programming languages [10]. The purpose of this approach is to strengthen the network and make it more diverse, with a view to avoiding a single client dominating the network in order to remove single points of failure. We extend this concept to the smart contract layer itself.

Our contributions are as follows.

- We introduce the notion of dissimilar redundancy for DeFi protocols
- We provide the first implementation of a protocol for dissimilar redundancy for a DeFi protocol¹
- We evaluate the protocol on a smart contract auction system implemented in both Solidity and Vyper, verify that a fuzzing approach would be able to detect

¹ <https://github.com/danhper/smart-contract-dissimilar-redundancy>.

purposefully introduced bugs, and provide the costs in USD of using a protocol for dissimilar redundancy

Outline

We set out the necessary background in Sect. 2, provide our methodology in Sect. 3, evaluate it in Sect. 4, consider related work in Sect. 6 and conclude in Sect. 7.

2 Background

In this section, we provide the necessary background regarding smart contracts, their potential vulnerabilities, as well as the cost of their execution.

2.1 *Smart Contracts*

Smart contracts are program objects that are native to blockchains. In the context of Ethereum, such smart contracts require that the underlying blockchain is a transaction-based state-machine. State changes occur on a blockchain through transactions, which are atomic: they either succeed, where the state is updated, or fail, where the state of the blockchain remains unchanged. For the approach we take below, atomicity is a crucial property of smart contracts as it provides that the blockchain cannot be left in an invalid or inconsistent state. Communication between two smart contracts occurs via message calls within the same execution context.

2.2 *Scaling Solutions*

Given the high cost of using Ethereum, many scaling solutions have emerged trying to solve this issue. At the heart of these approaches is taking computation off the layer-one blockchain, and instead performing this on a separate layer while using the layer-one blockchain as an anchor of trust. Such protocols are typically denoted layer two protocols, the name for a family of solutions that seek to allow applications to scale by processing transactions off the main blockchain.

In the context of Ethereum, one approach is to use rollups [11]. Rollups perform transaction execution away from the layer-one blockchain but store transaction data on-chain. In doing so, the security properties of layer-one are leveraged as an anchor of trust by the rollup approach, which is then able to perform transaction execution off-chain.

For a full summary of these approaches, see [13].

2.3 Smart Contract Vulnerabilities

Over the years, there have been many smart contract exploits often leading to significant losses of money [1, 17]. There are many different ways in which smart contracts can be exploited but on a very high level, attacks can be classified as “technical”, which means that an attacker can steal funds by exploiting a technical issue, such as a bug, in the contract, or “economical”, which means that the attacker can exploit a contract through incentives that might not be properly aligned [22]. For the purpose of this paper, we will focus on technical security of smart contracts. Within technical security itself, there is also a myriad of different potential issues, such as contract vulnerable to re-entrancy [19] or flash loan attacks [18]. However, another very common way in which contracts fail is simply logical bugs. Such logical bugs have cost millions to many smart contract projects. For example, Compound was recently victim of such an incident [3], that cost the protocol over 90 million USD. The error was as simple as a greater than symbol that should have been greater or equal. This is the class of bugs where dissimilar redundancy is the most promising.

3 Methodology

3.1 Overview

We now turn to how we use the approach of dissimilar redundancy in the context of Ethereum. At the centre of our approach is the use of a proxy architecture pattern. With a proxy pattern, all message calls to a contract \mathcal{C} first go through a proxy contract \mathcal{P} that serves to direct the message calls to contract \mathcal{C} . With this pattern, contract \mathcal{C} contains the actual implementation logic while contract \mathcal{P} provides a storage layer. At present, a common use of this pattern is to provide contract upgradeability [2, 16]: while contracts cannot be directly upgraded once deployed, upgradeability can be mimicked by changing where contract \mathcal{P} delegates calls to from contract \mathcal{C} to contract \mathcal{C}_1 .

We expand on this pattern: in our pursuit of dissimilar redundancy, we allow \mathcal{P} to delegate to multiple implementations at once. This is in contrast to the standard pattern which only permits delegation to a single implementation at a time. In a nutshell, our proxy contract \mathcal{P} sequentially calls two different implementations—supposedly identical in logic— \mathcal{C}_1 and \mathcal{C}_2 and ensures that the data returned by function calls to each implementation as well as return values from an arbitrary number of *checks* provided by the contract developer match. In this context, a check is a call to a contract’s function of which the return value is deemed relevant to the function called. For example, in the context of an ERC-20 token [21], the developer might want to add `balanceOf(from)` and `balanceOf(to)` as a check for `transferFrom(from, to, amount)`. The proxy will then call `balanceOf`

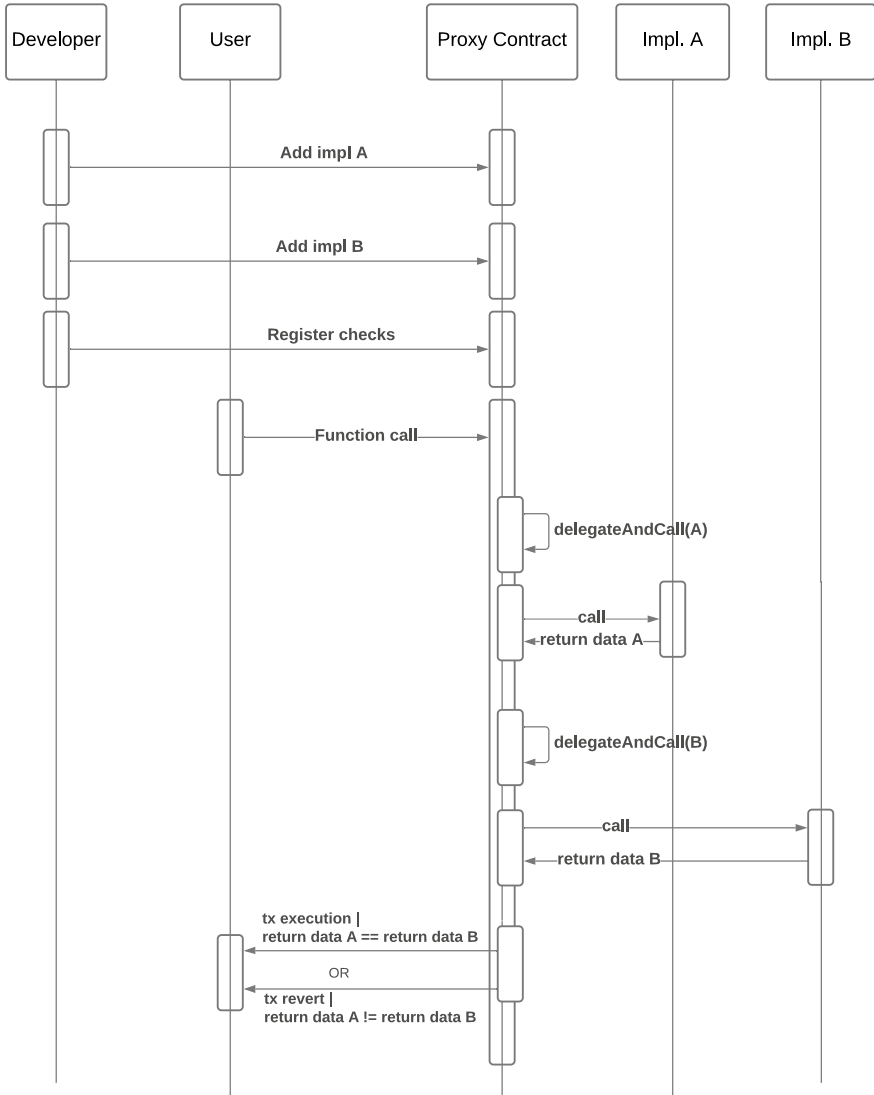


Fig. 1 Overview of our dissimilar redundancy framework

twice after each call to `transferFrom` and ensure that the results are consistent among the implementations.

Although the overall idea is straightforward, the actual implementation requires several technical difficulties to be overcome. We show an overview of the framework in Fig. 1.

3.2 The Technical Challenges

Calling implementations with the same state The first difficulty is that our approach requires both implementations to be called with the same initial state, sequential contract calls would typically modify this state. To overcome this, all state changes made by a call to an implementation need to be rolled back before we call the next implementation. We leverage the atomic nature of Ethereum’s calls to achieve this: if a transaction raises an error, the state reverts to the initial state.

In our approach, instead of the proxy directly delegating to an implementation, it first delegates to *itself*, passing in the call data as an argument along with the implementation to call and the checks to perform. In this delegated call, the proxy then delegates to the implementation, executes all the checks and combines their result in a single hash value. It then reverts the execution to rollback the changes made by the implementation and returns the checks hash as the revert data. The only exception to this is the call to the last implementation, where it returns the checks hash normally instead of reverting, to persist the changes. We provide a high-level overview of the delegation logic in Algorithm 1. Low-level implementation details can be found in our open-source implementation.

Check encoding The second difficulty concerns how the checks to be performed after each execution should be encoded. A check is a call to a contract that needs to be consistent after each execution. To make the proxy retain the interface of a proxied implementation, we must pre-register the checks with the proxy, rather than specifying the checks with each call. A naive implementation would permit the developer to register a function with pre-encoded arguments to be called after any call. However, such an implementation would have severe limitations. Pre-encoding arguments makes calls such as the one described above for `balanceOf` impossible, as these depend on call data and transaction information—this information is only available at runtime. Since these calls need to be well targeted for them to be effective and find potential discrepancies among implementations, such an approach would not be viable.

Instead, we implement an approach which achieves the following objectives:

1. the registration of per-function checks, without pre-encoded arguments. For example: `transferFrom` and `approve` could have a different set of checks registered.
2. call data and transaction information should be accessible during the checks

The first objective is easily achieved by storing a mapping from function signature to checks and retrieving the relevant checks depending on the function signature called by the current call to the proxy. The second objective is more challenging, as the developer must be able to register checks upfront that rely on information only available when the function is executed. To allow for this, rather than registering checks by passing in the arguments themselves, we design a simple byte encoding for the arguments that allows abstract arguments, registered with the checks, to be

Algorithm 1 Dissimilar redundancy framework call delegation

```

function CALLDELEGATE(impl, data, checks, isLast)
  (ok, retData) ← DELEGATETO(impl, data)
  checkResults ← RUNCHECKS(checks)
  checksHash ← HASHCHECKS(checkResults)
  if isLast then
    return (ok, retData, checksHash)
  else
    revert (ok, retData, checksHash)
  end if
end function

function REDUNDANTCALL(implementations, data)
  n ← LENGTH(implementations)
  signature ← GETSIG(data)
  checks ← GETCHECKS(signature)

  for i ← 0, n - 1 do
    impl ← implementations[i]
    last ← i == n - 1

    callData ← ENCODE(CallDelegate, impl, data, checks, last)
    (_, delegateRet) ← DELEGATETO(this, callData)
    (ok, retData, checksHash) ← DECODE(delegateRet)

    if ISDEFINED(previousOk) then
      ASSERT(previousOk == ok)
      ASSERT(previousRetData == retData)
      ASSERT(previousChecksHash == checksHash)
    else
      previousOk ← ok
      previousRetData ← retData
      previousChecksHash ← checksHash
    end if
  end for

  return (ok, retData)
end function

```

mapped to the concrete arguments that are computed when executing the checks. For example, an abstract argument could be “the first argument of the current call” or “the sender of the current transaction”. When executing the checks, the proxy will map these to the actual value of the first argument or the sender of the transaction, and encode these when calling the check function.

In Fig. 2, we show how we encode abstract arguments to register the checks. The first four bytes are, as for regular calls, the signature of the function to be called. The next byte is the number of arguments to pass to the function. Each argument can then be one of three types: a static argument, a call data argument or an environment

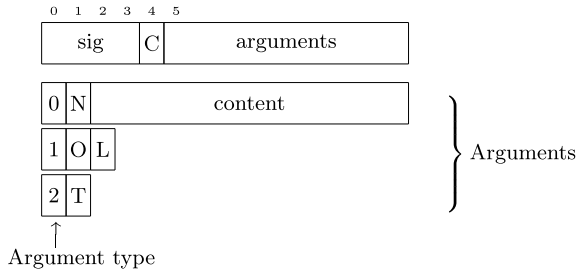


Fig. 2 Encoding for registering checks. **C** is the number of arguments. The three types of arguments are shown. Type 0 represents a static argument and **N** is the length of the static content. Type 1 represents call data argument where **O** and **L** are the offset and length to retrieve from call data. Type 2 represents environment argument and **T** is the type of environment variable (e.g. `msg.sender`) to use

argument. These types determine how the concrete argument will be computed when the contract is called:

- A static argument: simply passed through to the function as a concrete argument
- A call data argument: the given bytes from the transaction call data are extracted and passed as an argument
- An environment argument: looks up the concrete argument in the current transaction. The argument to be looked up in the environment, e.g., the sender of the transaction or the current block timestamp is specified by a single byte in the abstract argument.

Once all the abstract arguments are converted into their concrete counterpart, they are encoded along with the function signature and the check can be performed.

This encoding provides enough flexibility to perform a wide variety of different checks.

Toy example We show an example of such checks in Fig. 3, where we register three different checks for the `transferFrom` function of an ERC-20 token. The two first checks are for the balance of the first argument (the address sending tokens) and the second argument (the address receiving tokens) of the `transferFrom`. In both cases, we extract these arguments from the call data. The third check is for the allowance and also uses the first argument but also the sender of the transaction, as their allowance is expected to decrease after a successful call to `transferFrom`.

Now that we have established the methodology behind this approach to implementing dissimilar redundancy, we turn to an evaluation.

Fig. 3 Example checks registration for an ERC-20 token transferFrom function

```

tokenProxy.registerCheck(
    sig["transferFrom"],
    tokenProxy,
    encode_args(
        Token,
        "balanceOf",
        [(ArgumentType.CallData, (4, 32))]
    )
)

tokenProxy.registerCheck(
    sig["transferFrom"],
    tokenProxy,
    encode_args(
        Token,
        "balanceOf",
        [(ArgumentType.CallData, (36, 32))]
    )
)

tokenProxy.registerCheck(
    sig["transferFrom"],
    tokenProxy,
    encode_args(
        Token,
        "allowance",
        [(ArgumentType.CallData, (4, 32)),
        (ArgumentType.Env, EnvArg.Sender)]
    )
)

```

4 Evaluation

To evaluate our solution, we use a smart contract implementing a simple auction system. The rules of the auction system are as follows:

1. A seller starts an auction with an NFT of their choice and sets an end time
2. The NFT is transferred to the auction contract
3. Any user can bid in the auction and the bid must be strictly greater than the previous highest until the auction ends
4. After the auction ends, anyone can “finalize” the auction, which will either transfer the NFT to the winner of the auction or transfer it back to the owner if there were no bids

We implement the auction contract with the same behavior in both Solidity and Vyper but purposefully introduce a couple of implementation bugs in one of the two contracts. We then check whether these bugs would be detected by fuzzing the contract and causing the proxy to fail due to inconsistencies in the evaluation results.

In particular, we introduce two bugs to the Vyper version of the contract. First, rather than checking that the bid is strictly greater than the previous one, we check that the bid is greater or equal. This means that in this particular case, the Solidity

```

auction_proxy.registerCheck(
    sig["finalize"],
    nft_collection.address,
    encode_args(
        NftCollection,
        "ownerOf",
        [(ArgumentType.Static, ("uint256", NFT_ID))]
    )
)

@given(bids=st.lists(st.tuples(st.integers(min_value=0),
    addresses()))))
def test_bid(bids):
    with ensure_consistent():
        for value, account in bids:
            auction_proxy.bid({"from": account, "value": value})

@given(bids=st.lists(st.tuples(st.integers(min_value=0),
    addresses()))))
def test_finalize(bids):
    with ensure_consistent():
        for value, account in bids:
            auction_proxy.bid({"from": account, "value": value})
            chain.sleep(3600)
            auction_proxy.finalize()

```

Fig. 4 Testing code using dissimilar redundancy

version will revert the transaction while the Vyper one will successfully execute. Second, we omit the case where there were no bidders. As a result, both versions will successfully execute but for the Vyper implementation, the ownership of the NFT will still be the auction contract in the case there were no bidders.

4.1 Development-Time Testing

We first test our code locally to show how our approach can make bug-detection at development time significantly easier.

To test our code, we use Python in combination with the Brownie framework² for testing and the Hypothesis library [14] to generate test cases. We show the most important part of the code we use to our auction contract in Fig. 4. We note that the `ensure_consistent` context manager is implemented as part of our tooling and will only fail if a call reverts because implementations did not behave similarly.

In the tests, we first register a check for `finalize` that will look up the owner of the NFT that was on sale. For testing both `bid` and `finalize`, we generate random bids, where a bid is an account and value (price to pay for the auction) pair. For `bid`,

² <https://eth-brownie.readthedocs.io/>.

```
Falsifying example: test_bid_consistency(bids=[
    (1, <Account '0x66aB6...5871'>),
    (1, <Account '0x66aB6...5871'>)],
)
=====
FAILED tests/test_auction.py::test_bid -
brownie.exceptions.VirtualMachineError: revert: all
implementations must return the same success
```

(a) Failing test case for the `bid` function showing a failure after two bids with the same value were placed

```
Falsifying example: test_finalize_consistency(
    bids=[]
)
=====
FAILED tests/test_auction.py::test_finalize -
brownie.exceptions.VirtualMachineError: revert: all
implementations must return the same checks
```

(b) Failing test case for the `finalize` function showing a failure when no bids have been placed

Fig. 5 Failing tests when fuzzing the auction contract with a correct and an incorrect implementation

we only execute all the bids while for `finalize`, we also ensure that the auction is ended and execute `finalize`.

Using this approach, trying to fuzz the contract requires performing differential fuzzing on the different implementations of the contract logic registered by the proxy. This makes it possible to easily identify the cases where the two implementations do not behave in the same way.

In Fig. 5, we show the results of these tests. Note that we fix the first failing test before proceeding to the second one.

For the `bid` function, the test framework correctly outputs a failure when two bids are placed with the same value in a row. The failure scenario is clear from the test output, as the two bids of the input both have a value of 1. The error message mentions that all implementations should return the same “success”, which means that one of the implementations successfully executed (the bug-ridden version), while the other failed to execute.

The test for the `finalize` function also correctly fails with an example containing no bids. This is consistent with the bug in the Vyper version of the Solidity which does not transfer back the token properly to its original owner. The failing test also mentions that all implementations must return the same checks which means that all implementations had the same success status (they all succeeded in this particular case) but the checks did not return the same value. Indeed, a check was registered to look up the NFT owner.

Overall, with this example, we have seen that with only a few lines of code, it was possible to have extensive coverage of the tested function that is able to automatically find discrepancies among implementations and indicate to the developer the test cases that would yield different results.

4.2 Real-World Deployment

An important strength of our approach is that it is possible to utilize the two implementations not only at development and testing time but also after the contract is deployed, ensuring that all the transactions executed will always be consistent across the different implementations provided. To demonstrate how this would perform on a real-world blockchain, we deploy our auction and its two implementations on the Polygon main network, ensure that the calls that would trigger revert correctly, and measure the cost overhead of our approach. We provide a list of all the deployed contracts in Table 1.

To be able to give comparable results, we deploy our proxy using the Solidity and Vyper implementations, as well as a standalone version of each implementation. During our interactions with the contracts, we maintain a fixed gas price of 30 Gwei, which at the time of writing was enough for near instantaneous inclusion in a Polygon block.

We summarize the cost nominated in US dollars of all the interactions with our auction contracts in Table 2. We split the costs of the first and the subsequent bids,

Table 1 Contracts deployed on Polygon mainnet. “Sol” are Solidity contracts. “Vy” are Vyper contracts. “Proxied” are contracts used by the proxy. “Standalone” are contracts interacted with directly

Name	Address
Auction proxy	0xAd837BDD116C14aA82311Db7D1879C7cDDCfd283
Auction (Sol, proxied)	0xdb85f3DB2aA6E5e294485972ABE921be188b6A37
Auction (Vy, proxied)	0x9FD31161360B5E772f2b9C469D4A35E679273Dbf
Auction (Sol, standalone)	0xE575CCb0213393eBFc9258013af1c43e9E416544
Auction (Vy, standalone)	0xEe164319fE07127Efc8fd8b3e99ea736F8c955E

Table 2 USD cost (We use the December 12th 2021 price of 2.15 USD per MATIC token, Polygon’s native token used to pay for gas fees) of calling different functions of the auction contract with and without dissimilar redundancy proxy. The gas price is fixed to 30 Gwei

	Start	First bid	Subsequent bid	Finalize
Proxied	0.0229	0.0092	0.006	0.0144
Solidity	0.0094	0.0045	0.0029	0.0052
Vyper	0.0108	0.0045	0.0029	0.0071

since the first bid allocates storage, using more gas than subsequent ones. Since the proxy is using both the Solidity and the Vyper implementation, a lower bound for the cost is the sum of the cost of each individual implementation. The difference between the sum of these costs and the cost of calling the proxy is the overhead of the proxy itself, including the cost of calling checks after each call and checking consistency among results. In our example, only `finalize` has a check registered, to check for the owner of the NFT after execution. For the calls to `start` and `bid`, respectively about 1.2% and 2.2% of the total cost is part of the proxy overhead, the rest of the cost being used by the actual underlying functions. For `finalize`, the overhead is understandably higher as a check is registered. Indeed, about 14.5% of the cost is used by the proxy itself.

We also check that a transaction that would cause our correct and our buggy implementation to diverge would correctly revert. To do so, we bid using subsequently twice the same amount, which fails for the proxied version (tx [0x5f840a...6dc334c](#)) by returning the same error as the one we saw during the tests. As a sanity check, we try the same sequence of bids on the standalone implementations and the Solidity version reverts correctly (tx [0x406ad7...7759673](#)) while the Vyper one succeeds (tx [0x888189...7df41e8](#)).

5 Limitations

While our approach comes with strong benefits in terms of reliability, it has some limitations. In this section, we will go over these limitations and provide details on how some of them could be tackled.

5.1 *Transaction Fees*

As we have seen in the previous section, this approach will always at least double the cost of transactions and potentially increase it further if many calls need to be performed. There is not any direct way to prevent this issue or improve it significantly at the implementation level. This means that using such an approach on an expensive network, such as Ethereum, is almost infeasible, as the price increase for transactions would likely be unacceptable for many users. However, more and more layer two solutions are being developed, with transaction fees orders of magnitude lower than what can be seen on Ethereum mainnet. This is for example the case of Polygon mainnet, which we used for the evaluation earlier. As we can see in Table 2, although the costs have been doubled, this represents an increase in the order of a cent in the worst case. As Ethereum is moving towards a layer 2 future [5] and transaction fees become negligible, the fee overhead of our approach might become less and less of a concern.

5.2 *Development Cost*

Another obvious limitation is the development cost of another implementation of the same contract. However, there are several arguments why this might not be a critical issue.

First, such an approach has been seen with Ethereum clients, which are vastly larger in terms of complexity and code base than most protocols built on top of smart contracts. In a similar way that the Ethereum Foundation distributes funds to help external teams maintain clients, a well established protocol could potentially operate similarly to have alternative implementation maintained.

Second, given the often enormous amount of money at stake, a significant part of the smart contract development goes into testing rather than simply implementing the contract. For example, Uniswap V3 [20] has almost twice more test code than actual implementation. Our approach being highly complementary to regular testing, it could be integrated as part of the development process as yet another way to reduce the number of potential bugs or vulnerabilities in the contract.

5.3 *Storage Layout*

Another limitation of our approach is that all the implementations must use exactly the same storage layout. The proxy contract will be storing all the states of the contract, so all implementations must read at the same location in storage to be able to retrieve the information they are looking for. One of the main drawbacks is that it imposes some rigidity on alternative implementations, which might make it less likely to try implementing the logic in a different way and potentially replicate bugs in multiple implementations. Another issue is that this makes it hard to combine implementations written in Solidity and Vyper. The two languages use mostly the same mapping from state variable to Ethereum storage slot but for mappings, they use a very slightly different way to compute the storage slot which makes it impossible to use this approach for any contract using mappings. However, this issue is trivial to fix, as it would only require a very minor, albeit backward incompatible, change in the Vyper compiler.

6 **Related work**

We first present how a similar approach has been helpful for Ethereum clients and then present some related research discussing differential fuzzing.

6.1 *Ethereum Clients*

From early on, the Ethereum network has been running using different client implementations. The goal of having multiple clients has always been to make the network more robust and ensure that it does not fail in case there would be a bug, a vulnerability, or an avenue for a potential denial-of-service (DoS) attack in one of the implementations. In the case of Ethereum, this allows for several failure modes. If one of the implementations had a bug that would make it crash on certain transactions, the network would continue to operate with other implementations that do not contain this bug. This would be similar in the case of a DoS attack only effective on one type of implementation. On the other hand, if a bug or exploit would result in a different state transition after executing a transaction, it would create a network split where the victim implementation would only manage to reach consensus with nodes using the same implementation. This is riskier than the previous case but remains safe as long as exchanges and other entities bridging on-chain activity to off-chain assets ensure all implementations are in a consistent state before accepting to process a transaction. Overall, this diversity of clients has been very beneficial to Ethereum, despite the high maintenance cost, and has allowed it to operate smoothly for over 6 years.

6.2 *Differential Fuzzing*

Having two implementations that should behave in the same way allows to perform differential fuzzing: fuzzing both implementations trying to look for cases where they would behave differently. This technique has already been used in multiple domains such as cryptography [4], programming languages [6], and blockchain consensus [9, 12]. A recent work leveraging differential fuzzing to find bug in Ethereum clients [23] has managed to find not only most known consensus bugs but also two new ones, including a bug that led to a fork in the consensus due to only part of the full nodes in the network having upgraded to the latest version [7]. Overall, this shows the potential of differential fuzzing and how it can be useful for finding bugs and zero-day exploits.

7 Conclusion

We have argued that the high financial stakes in the context of DeFi merit an approach to program redundancy inspired by avionics: the utilization of dissimilar redundancy. Through implementing the same program logic more than once, ideally with different programming languages and even by different engineering teams, and then using an on-chain execution logic that ensures that the dissimilar implementations must *agree* before the on-chain state can update, redundancy is brought into the smart contract

ecosystem. Such redundancy should serve to make smart contracts, and DeFi as a whole, less vulnerable to exploits from implementation bugs.

We hope that this paper can offer one step on the path to a more robust and secure DeFi. As in avionics, in DeFi, the stakes are high, and the risks real.

Acknowledgements The authors would like to thank the Ethereum Foundation for their financial support.

References

1. Atzei, N., Bartoletti, M., & Cimoli, T. (2017). A survey of attacks on ethereum smart contracts (sok). In *International Conference on Principles of Security and Trust* (pp. 164–186) Springer.
2. Barros, G., & Gallagher, P. (2019). Eip-1822: Universal upgradeable proxy standard (uups). <https://eips.ethereum.org/EIPS/eip-1822>.
3. Bloomberg. (2021). Defi platform mistakenly sends \$89 million; ceo begs return. <https://www.bloomberg.com/news/articles/2021-10-01/defi-platform-mistakenly-sends-89-million-ceo-begs-its-return>.
4. Brubaker, C., Jana, S., Ray, B., Khurshid, S., & Shmatikov, V. (2014). Using frankencerts for automated adversarial testing of certificate validation in ssl/tls implementations. In *2014 IEEE Symposium on Security and Privacy* (pp. 114–129) IEEE.
5. Buterin, V. (2021). Endgame. <https://vitalik.ca/general/2021/12/06/endgame.html>.
6. Chen, Y., Su, T., Sun, C., Su, Z., & Zhao, J. (2016). Coverage-directed differential testing of jvm implementations. In *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 85–99).
7. Copeland, T. (2021). Bug impacting over 50% of ethereum clients leads to fork. <https://www.theblockcrypto.com/post/115822/bug-impacting-over-50-of-ethereum-clients-leads-to-fork>.
8. CryptoSec. (2021). Comprehensive list of defi hacks and exploits. <https://cryptosec.info/defi-hacks/>.
9. Ethereum. (2019). Evm lab utilities. <https://github.com/ethereum/evmlab>.
10. Ethereum. (2021). Ethereum nodes and clients—client diversity. <https://ethereum.org/en/developers/docs/nodes-and-clients/client-diversity/>.
11. Ethereum. (2021). Layer 2 rollups. <https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/>.
12. Fu, Y., Ren, M., Ma, F., Shi, H., Yang, X., Jiang, Y., Li, H., & Shi, X. (2019). Evmfuzzer: detect evm vulnerabilities via fuzz testing. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (pp. 1110–1114).
13. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., & Gervais, A. (2019). Sok: Off the chain transactions. *IACR Cryptol. ePrint Arch.*, 2019, 360.
14. MacIver, D. R., Hatfield-Dodds, Z., et al. (2019). Hypothesis: A new approach to property-based testing. *Journal of Open Source Software*, 4(43), 1891.
15. Mutual, N. (2021). A decentralized alternative to insurance. <https://nexusmutual.io/>.
16. Palladino, S. (2019). Eip-1967: Standard proxy storage slots. <https://eips.ethereum.org/EIPS/eip-1967>.
17. Perez, D., & Livshits, B. (2021). Smart contract vulnerabilities: Vulnerable does not imply exploited. In *30th USENIX Security Symposium (USENIX Security 21)* (pp. 1325–1341). USENIX Association. <https://www.usenix.org/conference/usenixsecurity21/presentation/perez>.
18. Qin, K., Zhou, L., Livshits, B., & Gervais, A. (2021). Attacking the defi ecosystem with flash loans for fun and profit. In *International Conference on Financial Cryptography and Data Security* (pp. 3–32). Springer.

19. Rodler, M., Li, W., Karame, G. O., & Davi, L. (2019). Sereum: Protecting existing smart contracts against re-entrancy attacks. In *Proceedings of 26th Annual Network & Distributed System Security Symposium (NDSS)*. <http://tubiblio.ulb.tu-darmstadt.de/111410/>.
20. Uniswap. (2020). Uniswap. <https://app.uniswap.org/#/swap>
21. Vogelsteller, F., & Buterin, V. (2015). Eip-20: Erc-20 token standard. <https://eips.ethereum.org/EIPS/eip-20>.
22. Werner, S. M., Perez, D., Gudgeon, L., Klages-Mundt, A., Harz, D., & Knottenbelt, W. J. (2021). Sok: Decentralized finance (defi). *CoRR*, abs/2101.08778. <https://arxiv.org/abs/2101.08778>.
23. Yang, Y., Kim, T., & Chun, B. G. (2021). Finding consensus bugs in ethereum via multi-transaction differential fuzzing. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)* (pp. 349–365). USENIX Association. <https://www.usenix.org/conference/osdi21/presentation/yang>.
24. Yeh, Y. (1996). Triple-triple redundant 777 primary flight computer. In *1996 IEEE Aerospace Applications Conference. Proceedings*, 1, 293–307. <https://doi.org/10.1109/AERO.1996.495891>.

DeFi Survival Analysis: Insights into Risks and User Behaviors



Aaron Green, Christopher Cammilleri, John S. Erickson, Oshani Seneviratne, and Kristin P. Bennett

Abstract We propose a decentralized finance (DeFi) survival analysis approach for discovering and characterizing user behavior and risks in lending protocols. We demonstrate how to gather and prepare DeFi transaction data for survival analysis. We demonstrate our approach using transactions in AAVE, one of the largest lending protocols. We develop a DeFi survival analysis pipeline which first prepares transaction data for survival analysis through the selection of different index events (or transactions) and associated outcome events. Then we apply survival analysis statistical and visualization methods such as median survival times, Kaplan–Meier survival curves, and Cox hazard regression to gain insights into usage patterns and risks within the protocol. We show how by varying the index and outcome events, we can utilize DeFi survival analysis to answer three different questions. What do users do after a deposit? How long until borrows are first repaid or liquidated? How does coin type influence liquidation risk? The proposed DeFi survival analysis can easily be generalized to other DeFi lending protocols. By defining appropriate index and outcome events, DeFi survival analysis can be applied to any cryptocurrency protocol with transactions.

A. Green (✉) · C. Cammilleri · J. S. Erickson · O. Seneviratne · K. P. Bennett
The Rensselaer Institute for Data Exploration and Applications, Rensselaer Polytechnic Institute,
New York, US

e-mail: green12@rpi.edu

URL: <http://idea.rpi.edu>

C. Cammilleri

e-mail: cammic@rpi.edu

J. S. Erickson

e-mail: erickj4@rpi.edu

O. Seneviratne

e-mail: senevo@rpi.edu

K. P. Bennett

e-mail: bennek@rpi.edu

1 Introduction

The rapid growth in popularity of blockchain-based products like cryptocurrencies has brought with it a growth in the complexity of the blockchain ecosystem. Myriad new products are being developed and deployed on the numerous blockchains that now exist. Following the invention and adoption of these products by an increasing number of users, a new financial ecosystem has emerged: Decentralized Finance (DeFi). The world of DeFi, though young, is already comprised of hundreds, if not thousands, of products and services. One such product is known as a lending protocol. DeFi lending protocols offer a similar set of services as banks offer to consumers in the world of traditional finance. Users utilize the protocols to conduct *transactions*. For instance, a user of a DeFi lending protocol can take some cryptocurrency they own and deposit it into the protocol, then accruing interest on their account balance. These users can also take out loans through the protocol, much like a person can take out a loan from a bank.

Since these lending protocols are growing in size along with the cryptocurrency market as a whole, an obvious first question might be, “how and why are people using DeFi lending protocols?” There are multiple popular lending protocols, and their data streams are varied. But these data streams share a common structure; entities are conducting different types of transactions through time, each involving varying amounts and types of cryptocurrency. The fact that these entities (we will call them users, but they may be smart contracts) conduct transactions at irregular intervals and the many types of transactions makes DeFi lending streams challenging to understand. However, this complexity also exists in domains such as healthcare and commerce, so there are myriad tools available to analyze temporal data streams. In this paper, we demonstrate how to use one such tool, “survival analysis,” to gain insight into DeFi transactions.

Survival analysis models time-to-event data. Survival analysis is widely used in healthcare to understand the risk of death (or other events of interest) after treatment, but can be used more generally to analyze the time between any two events [5]. For example, it can be used to analyze the time between a user’s borrow transaction and a transaction to repay that coin. As in healthcare, the data is frequently *right-censored*: users are likely to have an outstanding loan at any given time when we stop observing the data stream. In this analysis, we demonstrate how to gather and prepare DeFi transaction data for survival analysis and apply survival analysis statistical methods such as Kaplan–Meier survival curves and Cox Hazard regression to understand usage patterns within a protocol. Survival analysis has been previously used to understand loan defaults in Centralized Finance (CeFi) [2, 6, 10], but applying this analysis directly to DeFi is not straightforward since DeFi varies significantly from CeFi.

In order to interpret the results of this first use of survival analysis in DeFi, we focus our analysis on looking solely at one lending protocol, developing generic survival analysis tools for transaction analysis, and examining the results of these tools for that protocol. These same tools can eventually be applied to other lending

protocols. The protocol chosen was AAVE.¹ At the time of this writing (March 14, 2022), AAVE is the second-largest DeFi lending protocol, with approximately \$8.42 billion worth of crypto-assets locked in the protocol according to DeFi Pulse.² We note, however, that survival analysis tools for DeFi transactions can be used to analyze and gain insight into any DeFi protocol that consists of transactions through time, including other lending protocols and exchanges.

This paper is organized as follows: in the methods section, we describe the AAVE data and the survival analysis methods used to study it. In the results section, we demonstrate the use provided by survival analysis to answer three different questions. We conclude with a discussion of the contributions of this work and promising directions for future work.

2 Methods

2.1 Data

The data used in this analysis comes primarily from The Graph,³ a service that indexes data from blockchains and allows for the querying of the indexed data. For the work presented here, we sought to combine data from the primary seven transaction types that AAVE records in order to give a comprehensive view of all transactions that have taken place in AAVEv2 [1] since its deployment on November 30, 2020. The data used in this analysis starts with the first transactions of the AAVEv2 protocol on November 30, 2020, and ends on January 6, 2022. The seven transaction types we've pulled from The Graph include deposits, redeems, borrows, repays, liquidations, interest-rate swaps, and reserve collateral usage toggling.

Combining these seven transaction types, we get a table that includes one transaction per row, totaling 847,798 transactions. Table 1 summarizes the number of transactions of each type and their mean and median values. There are some common features for each transaction, such as the user involved and the time the transaction was made. Aside from liquidations, the transactions also have one specific coin involved. Liquidations have two coins: a principal coin and a collateral coin. AAVE transactions in our dataset have used 54 different coins. We divide these coins into two types: stablecoin and non-stablecoin. A stablecoin is from a class of cryptocurrencies that attempt to offer price stability (typically in terms of USD), and that is backed by a reserve asset. The other types of coins in the dataset are non-stablecoins.

Deposits and redeems in AAVE function as one might expect deposits and withdrawals to function at a bank. A user can deposit a currency into the AAVE protocol, accruing interest on their deposit through time. Upon depositing a currency, AAVE

¹ <https://aave.com>.

² <https://defipulse.com>.

³ <https://www.thegraph.com>.

Table 1 Summary of transaction types in AAVEv2 data collected from November 30, 2020 to January 06, 2022

Transaction type	Occurrences	Mean value (USD)	Median value (USD)
Borrow	124,899	\$337,019.10	\$14,983.18
Collateral	220,046	NA	NA
Deposit	239,836	\$482,453.00	\$4783.75
Redeem	170,516	\$843,124.80	\$26,633.09
Repay	81,650	\$448,525.00	\$25,314.20
Swap	2937	NA	NA
Transaction type	Occurrences	Mean principal (USD)	Mean collateral (USD)
Liquidation	7,914	\$74,798.06	\$79,682.95

mints the user some corresponding interest-bearing aTokens, which represents how much of a reserve they've deposited into the lending pool. These aTokens can be redeemed through the protocol to functionally withdraw their previously-deposited currency from the lending pool.

Borrows and repays function as their names would suggest. It is important to understand that borrowing a currency in AAVE is governed by smart contracts. Anyone can borrow any amount of currency from the AAVE lending pool as long as they follow the criteria specified by the appropriate smart contracts. Users who borrow in AAVE use the currency they have deposited into the protocol as collateral. Not all currencies that can be deposited in AAVE are allowable as collateral. Additionally, users can choose which of their deposited assets they want to allow for usage as collateral. In order to qualify to borrow some asset, a user must have enough deposited assets in the system that are usable as collateral so that the loan would be over-collateralized. The extent of over-collateralization required depends on the specific currencies being used as collateral. More specific details about the requirements for borrowing in AAVE can be read about in the AAVE whitepaper [1].

Liquidations, the most complicated of the transactions in our data, have a lot of unique information in each transaction. When a user performs a liquidation, they are always liquidating the account of another user. This means there are two users recorded for each liquidation transaction: the user being liquidated (the *liquidatee*), and the user performing the liquidation (the *liquidator*). Additionally, whereas other transaction types only interact with a single currency at a time, liquidations involve two currencies. There is the principal currency that the liquidator is paying off and the collateral currency that the liquidator is buying.

Collateral and swap transactions are the simplest transactions. Each one is functionally just the toggling of a setting in a user's account. Collateral transactions are made when a user wants to toggle whether a deposited currency can be used as collateral for loans they've made or plan to make. Swap transactions allow users to switch loans between stable interest rates and variable interest rates for an individual currency.

2.2 Survival Analysis for DeFi

Survival analysis is a collection of statistical procedures for data analysis in which the outcome variable of interest is the time from an index event until the outcome event [5]. To apply survival analysis to AAVE, we must pick two events: the index transaction and the outcome transaction. The survival time is the elapsed time between the index and outcome transactions. If, for example, we want to understand how soon users borrow money after making a deposit, then the index transaction is the user’s deposit, and the outcome transaction is the first borrow the user makes thereafter. The data is right-censored since we can only analyze data until the final date in our data. If a deposit transaction has no matching borrow, it could be because the user never borrowed or because the user borrow had not occurred during the period of analysis. Survival analysis correctly analyzes right-censored data.

The power of DeFi results from the fact that it can be performed on the time duration between any desired pair of transactions analyzed by any of the widely used statistical survival analysis techniques. In Sect. 3, we demonstrate how we can address many different questions by changing the definition of the index and outcome transactions. For each analysis below, we define precisely the index and outcome transactions. Survival analysis estimates the survival function for each of these pairs of transactions. The survival function captures the probability of the outcome transaction *not* occurring through time. We utilize the `ggsurvplot` function from the `survminer` package in R to produce Kaplan–Meier curves, the most popular way to both estimate and visualize survival functions. For example, Fig. 1 shows

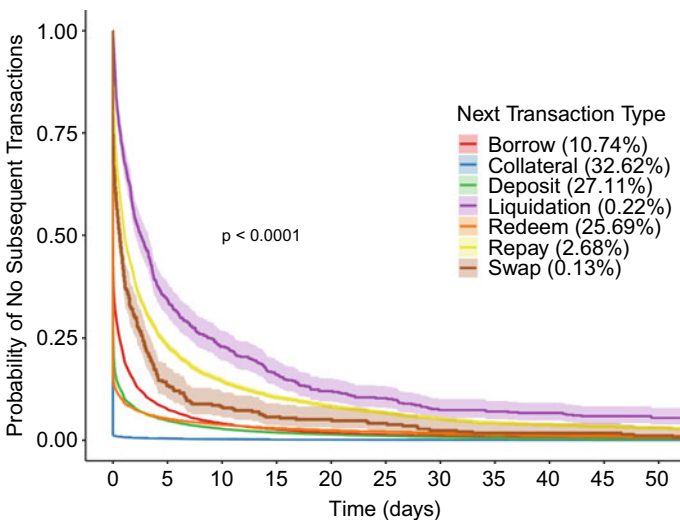


Fig. 1 Kaplan–Meier survival curves from a user’s deposits to their next transaction shows noticeable differences between which transactions users tend to make after a deposit

survival curves for each transactions type. They represent the time from the deposit to the first transaction of that type. If we are interested in how variables affect time to the outcome event, we utilize Cox regression (or proportional hazards regression). In Sect. 3.2, we utilize Cox regression to see if the coin type of borrows (stable or non-stable) is associated with faster liquidation rates.

3 Results

We use survival analysis to dissect the relationships between certain pairs of transaction types. In doing so, we demonstrate the effectiveness of survival analysis in visualizing and quantifying user behavior in AAVE. We explain how to transform the raw transaction data into forms suitable for survival analysis. Then we show different ways to use this data and survival analysis tools to uncover patterns of user behavior through the selection of different index events, the narrowing down of outcome events, and the separation of the data by other relevant features.

3.1 *What do Users do After a Deposit?*

We show that a Kaplan–Meier survival curve can provide a useful picture of how users behave after they deposit money into their accounts. Deposits are the natural first transaction for a user to make in a lending protocol, since before depositing any currency into an account, there aren't really other possible actions one can take. Thus, looking at how users behave after making deposits seems a natural place to begin our analysis.

To convert transaction data into survival data, we treat each deposit present in our data as an index event. This means we have 239,836 index events in the survival analysis. The outcome of each event occurs when the user makes their next transaction. The time difference between the deposit and the next transaction serves as the “survival” time for this analysis, and so if a user just makes a deposit and performs no subsequent transactions, that would manifest in our data as a deposit that has “survived” so far, and would be censored by time. For each deposit that is eventually followed by a subsequent transaction, we also record the type of transaction that follows. This gives data in the form given in Table 2.

Starting with the simplest survival analysis method, we used the R function `surv_median` to calculate the median time to the first transaction after a deposit. We computed the median time to any transaction and for each transaction type. We show these values in Table 3 ordered by median survival time. The percentage of the time this transaction type followed a borrow is provided as well. The median time between any two transactions is .017 hr. That means 50% of the time, a deposit by a user is followed by a transaction by that same user in less than 1.02 m. Note

Table 2 Survival Data from Deposits to Next Transaction

Time from index event (in hours)	Censored?	Next transaction type
0.0295	False	Deposit
16.373	False	Deposit
0.2765	False	Borrow
2.58	True	NA
⋮	⋮	⋮

Table 3 Median time from deposit to next transaction in hours with percentage occurring for each transaction type

Next transaction type	Median survival time in hours	Percentage
Any	0.017	100
Collateral	0.000 (<1 s)	32.62
Redeem	0.000 (<1 s)	25.69
Deposit	0.103	27.11
Borrow	0.231	10.74
Swap	15.108	0.13
Repay	25.488	2.68
Liquidation	61.053	0.22

that for this specific analysis, we do not consider the type of coin used in the second transaction (outcome event.)

From the median times to each transaction type following deposits, we can more clearly compare the magnitudes of elapsed time for different outcome transaction types. The speed at which users tend to engage in redeem and collateral transactions after a deposit is less than a second. The median survival to the next deposit and borrows is on the order of minutes. In contrast, many users take more than two days to repay or be liquidated after a borrow.

We can gain further analysis by plotting survival curves for this data split by the type of subsequent transaction to gain a more nuanced understanding of how quickly users make each type of transaction following a deposit. When we separate the survival curve by the next transaction type, the usage of survival analysis deviates slightly from its original intent. This is because when we separate the curve by subsequent transaction type, we are effectively separating the curve by a variable that only exists because the index event has “not survived,” i.e., the existence of a “next transaction type” means the outcome event has occurred, and thus we’ve cut out the censored transactions. However, we believe the value of the visualization remains intact.

Figure 1 shows seven different survival curves. As is the case for any survival curves, each curve begins at time 0 and has a probability of survival of 1. Then, as each curve progresses through time, the probability of “survival” drops in proportion to how many of the cases represented by each curve have reached their corresponding outcome event. If a curve drops quickly to a low survival probability, that means users tend to make that transaction type quickly after making a deposit.

Looking at Fig. 1, it is apparent that the most common transaction users make after a deposit is the collateral transaction. This makes sense, as one of the requirements in AAVE for a user to take out a loan is that the loan is over-collateralized; according to the requirements of the protocol, the user cannot take out a loan without having any currency in their account that is marked as collateral. Still, the “survival” of a deposit until a collateral transaction is brief. The survival curve for collateral dips straight down and hits zero almost immediately, indicating that users don’t wait long before making collateral transactions.

3.2 *How Long Until Borrows are Repaid or Liquidated?*

Since survival analysis allows for flexible choices of index and outcome events, we turn our focus now to using borrows as the index events and the relevant transactions of repays and liquidations as the outcome events. Using only transaction-level data to analyze borrows makes it difficult to track a loan in its entirety. There is no end date to the loan. The user can make several borrows of a coin and then maintain the loan that accrues interest until they repay it in one or more repay transactions for that coin or until part or all of the loan is liquidated. When converting the transaction data into a form usable for survival analysis with borrows as the index events, we had to decide what was an appropriate outcome event. To avoid making assumptions about when a loan is totally repaid (either through liquidations or repay transactions), we define the outcome events as just the first repayment that a user makes or the first liquidation that is made for the borrowed currency. Thus, to be clear, the following analysis in Fig. 2 does not show how long it takes for loans to be totally repaid through repays or liquidations, but just how long it takes for them to *start* being repaid through either means.

We also choose to split the two curves by whether the borrowed coin is a stablecoin. The use cases for borrowing stablecoins versus non-stablecoins are quite different, so we hoped to see a drastic difference in the repayment schedules and liquidation tendencies for these different coin types, and indeed this is what we see in Fig. 2.

Clearly, from these survival curves, users tend to repay loans of stablecoins much more often than non-stablecoin loans. Almost all loans of stablecoins end up being repaid at least in part by the 402-day cutoff, which is as much data as we have. This is in stark contrast with the non-stablecoin borrows, for which only about 50% of all loans have seen even a single repayment. We see similarly contrasting behavior for the frequency that loans of each coin type are liquidated. Loans of stablecoins are liquidated significantly more often than loans of non-stablecoins. Unsurprisingly,

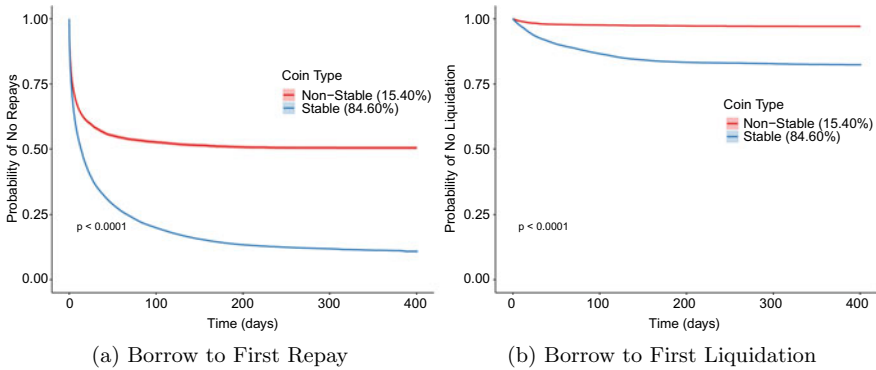


Fig. 2 Borrow to first repay and first liquidation analysis shows significant differences between stablecoins (red) and non-stablecoins (blue). Note that the time ranges for repay is smaller than that of liquidation

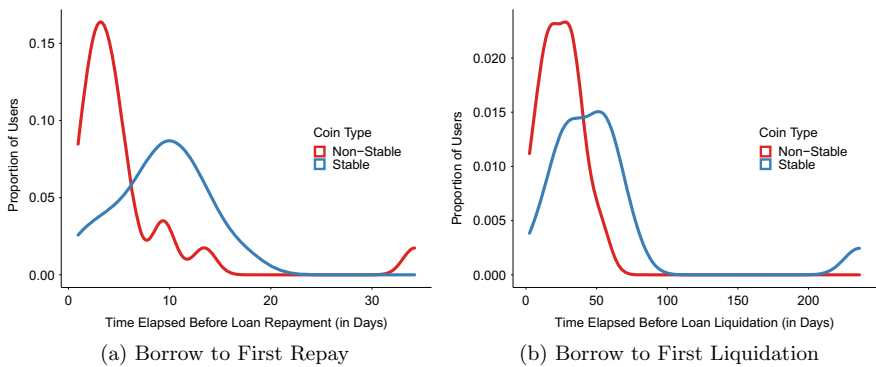


Fig. 3 Borrow to first repay and first liquidation analysis both show differences between stable and non-stablecoins

liquidations as a whole occur far less frequently than repayments, so the survival probability of loans relative to liquidations is much higher than loans to repayments, but the same patterns are present with respect to the type of coin being borrowed.

We can also use the survival data to create density plots for the median time to repayment or liquidation for each coin type, which provides additional clarity on exactly how long it takes for repayments or liquidations to occur. These median repayment and liquidation times can be seen in Fig. 3. From these, we can see that, despite fewer repayments and liquidations of non-stablecoins, the median times to these events are quicker than they tend to be for stablecoins. Most people make their first repayment of a borrowed non-stablecoin in five days or less, whereas for borrowed stablecoins, the median time until the first repayment is about ten days. The timelines for liquidations to occur are longer, with the highest proportion of liquidated borrows of non-stablecoins occurring between 20 and 30 days following the borrow,

and between 25 and 75 days for borrows of stablecoins. Still, the contrasting behavior between stablecoin and non-stablecoin borrows are similar for the time taken to repays and time taken to liquidations.

3.3 *How Does Coin Type Influence Liquidations?*

In the previous section, we saw the dramatic impact of coin-type of the principal on time to the first liquidation of a borrow. We hypothesize that the combination of the principal and the collateral may lead to further insight into the risk of borrows. Thus we further separate the borrow-to-liquidation data by factoring in what collateral was purchased and what principal types were specifically paid off by the liquidator. We perform the same index and outcome events as the prior liquidation analysis; only now do we analyze borrows associated with liquidation. This gives the curves seen in Fig. 4. Since we are splitting the curves by what principal and collateral were paid off and purchased at the time of the liquidation, all the curves do end up with a 0% probability of survival, similar to the curves in Fig. 1. Again though, we can still use the curves to gain insight into the relative riskiness of the principal:collateral combinations that people can have in their accounts. According to the log-rank test, the differences in the curves are statistically significant.

The definition of the outcome event in this analysis is quite different. To gain a more accurate picture of the liquidated user's account, we aggregated liquidation events to gain more information as to which coins the users have as collateral in their account. Even though each liquidation transaction only records one principal type and one collateral type, sometimes a user will be the subject of multiple liquidations in quick succession. It would be inaccurate to consider these liquidations as separate events; they really are all part of one bigger liquidation event. Thus, in our transaction data, if a user is liquidated multiple times in quick succession with no intermittent non-liquidation transactions, we aggregate them into one bigger liquidation transaction. The outcome event is the combined liquidation transaction, with the time being the first liquidation transaction. This lets us see whether there were multiple types of collateral and principal coins involved in the event. Thus, if a user has both stablecoins and non-stablecoins in their account as collateral, or if they've taken out loans of both stablecoins and non-stablecoins, we mark the collateral or principal, respectively, as "Mixed."

Table 4 shows an example of where we can use a Cox proportional hazard model to more effectively quantify the differences between survival probabilities of each type of principal:collateral combination leading to liquidations. For the Cox proportional hazard model, we need to choose one of the combinations of principal:collateral to which to compare the others, which tells us proportionally how risky the other types of principal:collateral combinations are relative to the benchmark combination. If we select the stable:stable combination as the benchmark, we get the quantification of risk via the `coxph` function from the `survminer` package as seen in Table 4.

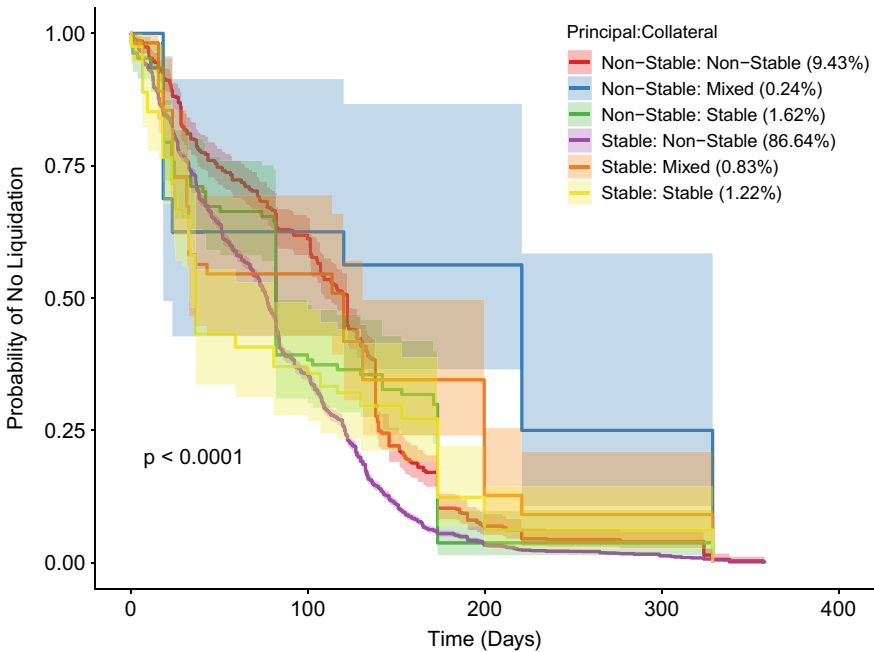


Fig. 4 Survival curves for different combinations of principal and collateral coin types

Table 4 Cox proportional hazards coefficients quantifying risk of liquidation. Bolded principal-collateral combinations have significant differences in risk of liquidation relative to stable:stable. Percentage of liquidation events indicates number of events of each type. Stable:stable constitutes 1.22% of liquidations

Principal: Collateral combination	Coefficient	p-value	Percentage
Stable:Non-Stable	0.24499	0.02903	86.64
Non-Stable:Stable	-0.08756	0.55238	1.62
Non-Stable:Non-Stable	-0.17044	0.14967	9.43
Stable:Mixed	-0.32461	0.06328	0.83
Non-Stable:Mixed	-0.74976	0.00617	0.24

The “Coefficient” column of this table indicates the relative riskiness of the loan types as compared to the benchmark type of stable principal and stable collateral. Negative coefficients are indicative of lesser risk, meaning that any of the principal:collateral combinations that have a negative coefficient are less likely to be liquidated than the stable:stable combination. The p-value tells the statistical significance of these coefficients, with lower p-values indicating that the corresponding coefficients were less likely to be generated by chance. The results show that the com-

binations of stablecoin principal with non-stablecoin collateral tend to be liquidated significantly sooner than stablecoin principal with stablecoin collateral loans. Borrowers with stablecoin principal with non-stablecoin collateral are much riskier since the price of the cryptocurrency in dollars is highly volatile. We see that liquidations for borrowers with mixed collaterals reduce the risk for both types of principals. There were no significant differences in time to liquidation compared with stable:stable combinations for the non-stable:stable and non-stable:non-stable combinations.

4 Related Work

With an over-collateralized loan, a borrower must post collateral, i.e., provide something of value as security to cover the value of the debt, where the value of the collateral posted exceeds the value of the debt. This way, collateralization simultaneously ensures that the lender (likely a smart contract) can recover their loaned value and provides the borrower with an incentive to repay the loan. The “health factor” (HF) is a custom threshold in lending systems. If the debt collateral falls below the HF (typically below 1), the debt position may be opened for liquidation. Then the liquidators can purchase the locked collateral at a discount and close the borrower’s debt position. Thus, leveraged positions are subject to liquidation when the debt becomes unhealthy, and a liquidator can repay the debt and benefit from a liquidation spread.

Given this novel form of automatic lending, a growing body of literature has studied liquidations on borrowing and lending platforms in DeFi. Qin et al. [9] have analyzed risk management provided by liquidators, acting on the protocol’s user accounts. They have measured various risks that liquidation participants are exposed to on four major Ethereum lending pools (i.e., MakerDAO, AAVE, Compound, and dYdX) and quantified the instabilities of existing lending protocols. They have illustrated that the commonly used incentive mechanisms tend to favor liquidators over borrowers, causing the problem of so-called over-liquidation, leading to unnecessary high losses for borrowers. The only recourse the borrowers have to avoid such liquidations is to monitor their loan-to-value ratio when the market changes quickly because even a random drop in market prices can result in a cascade of liquidations. If there are any drops in the market, it can lead to self-accelerating pressure to sell, which further causes more problems for a blockchain-based DeFi, such as network congestion that leads to steep gas costs. We witnessed such an event in the ETH market collapse of March 13, 2020⁴ that left some borrowers unable to react, despite imminent liquidations. It can be particularly bad for borrowers who get liquidated if market prices recover after a dip again, leaving them deprived of subsequent upward price participation. In general, regardless of market conditions, liquidations in DeFi are widely practiced, and related works such as Qin et al. [9] have quantified that

⁴ <https://coinmarketcap.com/historical/20200313>.

over the years 2020 and 2021, liquidators realized a financial profit of over 800M USD while performing liquidations.

Stablecoins play a significant role in liquidations, as they have several characteristics that are directly tied to liquidation mechanics. For example, a user may not want to sell the token collateral, which is usually in the form of a stablecoin, but instead hold it indefinitely as a means of passive income, which might exceed the cost of borrowing, making the transaction profitable. Early empirical evidence on the stability of crypto-backed loans with stablecoins has been studied by Kozhan and Viswanath-Natraj [7]. They specifically focused on the price volatility in the Maker-Dao protocol, which introduced the world's first decentralized stablecoin called Dai that is soft-pegged to the US Dollar, i.e., it uses a collateralized debt position mechanism to keep the price stable with respect to the US Dollar. They have analyzed how collateral stability increases peg stability and found a positive relationship between collateral risk and the price volatility of the stablecoin Dai.

The efficiency of lending pool liquidations has been studied by Perez et al. [8], in which they introduced a lending pool state model that is instantiated with historical user transactions observable in the Compound⁵ implementation deployed on Ethereum. Their model abstraction facilitates the observation of state effects of each interaction and investigates the latency of user liquidations following the under-collateralization of borrowing accounts. Similarly, Bartoletti et al. [3] provide an abstract formal state transition model of lending pools and prove fundamental behavioral properties, which had previously only been presented informally in the literature. Additionally, the authors examine attack vectors and risks, such as utilization attacks and interest-bearing derivative token risk.

As the demand for loans in crypto-assets grows, the borrowing interest rate goes up. In a bullish crypto market, speculators may be keen to borrow funds even if there is a high interest rate, in expectation of an appreciation in the assets of their leveraged long position as demonstrated by Xu et al. [13]. Such an environment is advantageous for lenders, resulting in higher yields to them. Compound and AAVE, two major DeFi lending protocols, have witnessed the borrow APY of the stablecoin USDC increasing from a low of 2–3% in May 2020 to as high as 10% in April 2021 (as of this paper writing in March 2022, the APY is hovering at 2% in Compound and AAVE, but other protocols such as Celcius, BlockFi and Nexo offer upwards of 8% APY).⁶ In a bullish market, the yield generated is incorporated in interest-bearing tokens, such as aTokens from AAVE analyzed in this paper. However, as was already noted, the wild fluctuations in the market result in unexpected liquidation events, as evidenced from this paper's results.

⁵ <https://compound.finance>.

⁶ https://defirate.com/usdc/?amount=100&symbol=USDC&term=365&rate_type=lend.

Most of the related works approach the issue of liquidation at a conceptual level or rely on aggregate flow data. In contrast, our paper uses transaction-level blockchain data to provide a more microscopic view on the issue combined with survival analysis techniques.

5 Discussion and Future Work

This work defines a pipeline for survival analysis of DeFi lending protocols which includes data aggregation, cleaning, converting to a data abstraction model, and performing powerful survival statistical analyses and visualizations to gain insights. Using AAVE lending data in three different scenarios, we demonstrate how to gain insights into user behaviors and loan risks by defining appropriate index and outcome events and then applying survival analysis.

Each scenario is characterized by distinct definitions of the index and outcome transactions in the survival analysis. We characterize user behaviors using survival analysis of AAVE users' next transaction. Our analysis of borrowed coin types shows the value of survival analysis for discovering factors contributing to events. Our survival analysis of borrows to repays and liquidations showed that borrows of stablecoins versus non-stablecoins exhibit very different characteristics. Users hold non-stable loans longer before the first repayment. But if they do repay, they tend to repay more quickly. We could get a deeper understanding of AAVE user behavior by taking a more refined look at the types of coins and transaction volumes, incorporating external factors such as coin prices in the survival analysis, and defining alternative index and outcome events. Using machine learning to create clusters that capture different behaviors (e.g., retail versus institutional investors), and then doing survival analysis could also be very illuminating. We leave these to future work.

This work represents just the first step in the use of survival analysis in DeFi. We note that these DeFi survival analysis techniques could be generalized to other DeFi lending protocols. DeFi survival analysis can be applied to any cryptocurrency protocol with transactions. Hazard analysis and other types of survival analysis and visualization methods could be used. As future work, we will prepare a toolkit for DeFi survival analysis with associated dashboards and demonstrate it on other DeFi Protocols.

One limitation of this research is that it does not address the rich DeFi ecosystem, which has many interacting protocols and coin prices. We are already exploring the use of more advanced Artificial Intelligence (AI) methods for the analysis of transaction data developed for commerce and health [4, 12] that could incorporate more aspects of the DeFi ecosystem. These could be used for segmenting users and predicting behaviors and prices. Early results analyzing AAVE transactions using Neural Temporal Point Processes are very promising [11]. DeFi represents an exciting new domain for AI research in transaction modeling since DeFi is a compelling use case, and all the datasets are public by definition.

The code used to generate the figures in this paper is available in a public GitHub repository.⁷

Acknowledgements This work was supported by the Center for Research towards Advancing Financial Technologies (CRAFT), the National Science Foundation (NSF), and the Rensselaer Institute for Data Exploration and Applications (IDEA). We would like to thank Roman Vakhrushev for his contributions to the liquidation analysis, Soumya Mishra, and Jaimin Vyas for contributions to early versions of the survival analysis.

References

1. AAVE Developers. (2020). AAVE protocol whitepaper V2.0. *Tech. Rep.*
2. Allen, L. N., & Rose, L. C. (2006). Financial survival analysis of defaulted debtors. *Journal of the Operational Research society*, 57(6), 630–636.
3. Bartoletti, M., Chiang, J. H. y., & Lafuente, A. L. (2021). Sok: lending pools in decentralized finance. In *International Conference on Financial Cryptography and Data Security* (pp. 553–578). Springer.
4. Bennett, K. P. (2019). Artificial intelligence for public health. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (pp. 1–2). <https://doi.org/10.1109/BIBM47256.2019.8983112>.
5. Clark, T. G., Bradburn, M. J., Love, S. B., & Altman, D. G. (2003). Survival analysis part i: basic concepts and first analyses. *British journal of cancer*, 89(2), 232–238.
6. Dirick, L., Claeskens, G., & Baensens, B. (2017). Time to default in credit scoring using survival analysis: a benchmark study. *Journal of the Operational Research Society*, 68(6), 652–665.
7. Kozhan, R., & Viswanath-Natraj, G. (2021). Decentralized stablecoins and collateral risk. *WBS Finance Group Research Paper Forthcoming*.
8. Perez, D., Werner, S. M., Xu, J., & Livshits, B. (2021). Liquidations: Defi on a knife-edge. In *International Conference on Financial Cryptography and Data Security* (pp. 457–476). Springer.
9. Qin, K., Zhou, L., Gamito, P., Jovanovic, P., & Gervais, A. (2021). An empirical study of defi liquidations: Incentives, risks, and instabilities. In *Proceedings of the 21st ACM Internet Measurement Conference* (pp. 336–350).
10. Roszkowska, P., & Prorokowski, Ł. (2013). Model of financial crisis contagion: A survey-based simulation by means of the modified kaplan-meier survival plots. *Folia Oeconomica Stetinensia*, 13(1), 22–55.
11. Shou, X., Gao, T., Subramanian, D., & Bennett, K. P. (2022). Multi-label event prediction in continuous time. Under review.
12. Shou, X., Mavroudeas, G., New, A., Arhin, K., Kuruzovich, J. N., Magdon-Ismail, M., & Bennett, K. P. (2019). Supervised mixture models for population health. In *2019 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)* (pp. 1057–1064). <https://doi.org/10.1109/BIBM47256.2019.8983339>.
13. Xu, J., & Vadgama, N. (2022). From banks to defi: the evolution of the lending market. In *Enabling the Internet of Value* (pp. 53–66). Springer.

⁷ <https://github.com/aaronmicahgreen/DeFi-Survival-Analysis-Insights-into-Risks-and-User-Behaviors>.

Gas Gauge: A Security Analysis Tool for Smart Contract Out-of-Gas Vulnerabilities



Behkish Nassirzadeh , Huaiying Sun , Sebastian Banescu ,
and Vijay Ganesh 

Abstract In recent years, we have witnessed a dramatic increase in the adoption and application of smart contracts in a variety of contexts. However, security vulnerabilities pose a significant challenge to the continued adoption of smart contracts. An important and pervasive class of security vulnerabilities that afflicts Ethereum smart contracts is the *gas limit DoS on a contract via unbounded operations*. These vulnerabilities result in a failed transaction with an “*out-of-gas*” error and are often present in contracts containing loops whose bounds are affected by end-user input. To address this issue, we present Gas Gauge, a tool aimed at detecting Out-of-Gas DoS vulnerabilities in Ethereum smart contracts. The Gas Gauge tool has three major components: The Detection Phase, Identification Phase, and Correction Phase. The Detection Phase component consists of an accurate static analysis approach that finds and summarizes all the loops in a smart contract. The Identification Phase component uses a white-box fuzzing approach to generate a set of inputs that causes the contract to run out of gas. Lastly, the Correction Phase component uses static analysis and run-time verification to predict the maximum loop bounds consistent with allowable gas usage and suggest appropriate repairs to the tool’s users. Each part of Gas Gauge can be used separately or all together to detect, identify and help repair contracts vulnerable to Out-of-Gas DoS vulnerabilities. Gas Gauge was tested on 1,000 real-world solidity smart contracts. When compared to seven state-of-the-art tools, we show that Gas Gauge is the most effective (i.e., has no false positives and false negatives) while being competitive in terms of efficiency.

B. Nassirzadeh (✉) · V. Ganesh
University of Waterloo, Waterloo, Canada
e-mail: bnassirz@uwaterloo.ca

V. Ganesh
e-mail: vijay.ganesh@uwaterloo.ca

H. Sun
East China University of Science and Technology, Shanghai, China
e-mail: ecustshy@foxmail.com

S. Banescu
Quantstamp, Munich, Germany
e-mail: sebi@quantstamp.com

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes
in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_9

Keywords Smart contract security · Blockchain · Ethereum · Static analysis · Dynamic analysis

1 Introduction

Smart contracts are one of the main applications of leading blockchains such as Ethereum [28]. Smart contracts are executable programs that allow building a programmable value exchange or a contract between various parties without the need for a trusted third-party. While smart contracts bring many benefits to the blockchain ecosystem, they suffer from various security vulnerabilities. Certain vulnerabilities can be exploited by attackers to steal funds from a smart contract, while others can cause funds to be locked indefinitely. One security issue plaguing Ethereum smart contracts is Out-of-Gas Denial of Service (DoS) vulnerabilities. Every operation in an Ethereum smart contract costs a certain amount of gas, a measurement unit for the amount of computational effort required to execute said operation or transaction, paid by the transaction initiation party. Each block comes equipped with an upper bound on the amount of gas that can be spent to compute all the transactions within that block. This is called the *Block Gas Limit*. Since a transaction cannot exceed one block, the transaction gas limit is also bound by the block gas limit [17]. One of the kinds of gas-related vulnerabilities is *DoS with Unbounded Operations*, also known as *Unbounded Mass operations* [13]. In this case, the execution of the transaction requires more gas than the block gas limit. As a result, The execution of one or more functions in a smart contract vulnerable to Out-of-Gas can be blocked indefinitely if Out-of-Gas conditions are not appropriately handled. This is the type of vulnerability that we focus on in this paper.

Many Ethereum wallets, such as Metamask [6], have a built-in mechanism to estimate the cost of a transaction statically, right before it is executed. However, there are certain cases when the gas estimation is incorrect or impossible to estimate. For example, if multiple operations are performed inside a loop traversing a dynamic array or mapping [1]. The Metamask Support website acknowledges this issue [4] and indicates that users should manually adjust the transaction gas limit according to one of the latest passing transactions for the smart contract function they are trying to call.

In order to address the above-mentioned problem, we present a tool, Gas Gauge, that automatically detects and suggests remediation for Out-of-Gas vulnerabilities in Ethereum smart contracts. We were motivated by two factors in designing Gas Gauge. First, to-date, the most widely used method for detecting and repairing Out-of-Gas vulnerabilities is manual security audits which cannot continue to scale with the ever-growing size, complexity, and the number of smart contracts. Second, as we show in this paper, existing automated methods based on static, symbolic, or run-time analysis are plagued by either high false negative rates or scalability issues. Our insight is that the best way to address this problem is to use an appropriate combination of static and dynamic analysis methods. Hybrid methods can outperform all other approaches

because, in order to detect these vulnerabilities, one needs to determine the loops or functions that can lead to an Out-of-Gas DoS (best done via static analysis), and then perform gas analysis to determine the exact point, e.g., loop iteration, when Out-of-Gas occurs (best done using appropriate dynamic analysis).

1.1 Contributions

The contributions we make in this paper are as follows:

- i. Design and implementation of a static analysis and run-time verification tool, Gas Gauge,¹ aimed at automatically detecting Out-of-Gas DoS vulnerabilities in Ethereum smart contracts. We implemented three techniques in Gas Gauge that are critical for identifying Out-of-Gas DoS vulnerabilities. The first technique is a static analysis method that identifies and summarizes loops in smart contracts. The second is a white-box fuzzing technique that triggers Out-of-Gas errors in smart contracts that contain publicly accessible functions with loops whose bounds are influenced by inputs (To the best of our knowledge, this feature is unique to Gas Gauge). The third is a method for identifying a threshold, as a function of input and state variables, at which Out-of-Gas errors can be triggered in a contract-under-analysis (To the best of our knowledge, this feature is also unique to Gas Gauge).
- ii. An extensive evaluation of the Gas Gauge against seven state-of-the-art tools: GASTAP/Gasol [2, 3], Madmax [13], MPro [27], Mythril [7], Securify 2.0 [25], Slither [11], and SmartCheck [23] on a benchmark of 1,000 real-world smart contracts. Our evaluation found that Gas Gauge outperforms these state-of-the-art tools. That is, Gas Gauge has zero false negative and false positive rates, while at the same time has a similar efficiency profile to the fastest tool in the set, SmartCheck. Further, none of these tools provide any support for repair, a feature that is essential for the industry.
- iii. A case study on a real-world application, Airswap, a peer-to-peer trading network for Ethereum used for trading of over \$20 million/week with current ETH value. We consulted with the engineers who manually audited Airswap and were informed that constructing the repair for the Out-of-Gas DoS vulnerability in Airswap took them several man-hours of work, while our Gas Gauge tool managed to automatically accomplish the same task on this real-world smart contract in under 10 m.

Our experimental evaluation reveals that Gas Gauge is useful, accurate, and outperforms other state-of-the-art tools that can be used to detect Out-of-Gas vulnerabilities. We tested our tool on 1,000 real-world smart contracts extracted from Etherscan,² one of the most popular Ethereum blockchain explorers. All these contracts

¹ <https://gasgauge.github.io/>.

² <https://etherscan.io/>.

were manually checked to ensure that they contain at least one loop since user-controlled loops are one of the main causes of *DoS with Unbounded Operations* [13]. Our tool uses run-time analysis to ensure no false positives, and our static analysis method ensures a zero false negative rate on the evaluated benchmark. Also, we performed a case study that showed that Gas Gauge could be utilized in real-world projects written in Solidity to save hours of manual work and millions of USD.

2 Background

Background on Smart Contracts: Ethereum is one of the leading blockchain platforms. It is a decentralized and open-source blockchain that contains millions of accounts and billions of USD in capitalization. Hence, it is one of the most prevalent underlying technologies for smart contracts [27]. Smart contracts handle transactions in a cryptocurrency called Ether. They are commonly written in the Solidity language, which is a Turing-complete programming language [10]. It then gets compiled to Ethereum Virtual Machine (EVM) bytecode instructions to be deployed on the blockchain. Unlike traditional software programs, a smart contract is publicly accessible, transparent, and immutable. Therefore, once a smart contract is deployed on the blockchain, it cannot be altered. Thus, if errors or vulnerabilities are found in a smart contract post-deployment, they cannot be fixed a posteriori by developers unless CREATE2 is used [19]. Several smart contract vulnerabilities have been discovered in recent years, of which Out-of-Gas DoS are among the common ones. A list of the most known smart contract vulnerabilities can be found on the SWC Registry website.³

Helper Tools: Developing an automatic gas analysis tool from Solidity source code requires a considerable implementation effort. Fortunately, many existing open-source tools make this task easier than otherwise. For example, we used the Slither [11] and Truffle Suite [9] as part of the Gas Gauge. Slither provides many useful APIs to collect information about a smart contract, such as data dependencies and function signatures. This is used to summarize the contract and extract the needed information for the other parts of Gas Gauge implementation. Also, the Control Flow Graph (CFG) generated by Slither is used to find the loops in the contract and their orders. Further, Truffle Suite is used to compile and deploy Solidity smart contracts to a test Ethereum network. This allows us to use different sets of inputs in the Identification Phase of the tool and use different threshold values in the Correction Phase while retrieving useful gas-related information such as gas used and gas left.

³ <https://swcregistry.io/>.

Whitebox Fuzzing: Nowadays, security vulnerabilities in software products can be found by two fundamental methods: Code inspection of binaries and black-box fuzzing. Blackbox fuzzing is a class of blackbox random testing that randomly mutates well-formed inputs to the program and then tests the program on the resulting data in order to trigger a bug [12]. Blackbox fuzzing is an effective method to test a program; however, it can have limitations. Low code coverage is one of the leading limitations of blackbox fuzzing resulting in missing security bugs [12]. An alternative approach to blackbox fuzzing is whitebox fuzzing. It is a type of automatic dynamic test generation, based on symbolic execution and constraint solving, intended for large applications' security testing [12].

2.1 *Out-of-Gas Denial of Service Vulnerabilities*

The gas fee has to be paid by the transaction initiation party before the execution starts. Since estimating the exact gas needed can be challenging, as described in Sect. 1, the transaction initiators can specify the maximum amount of gas they are willing to pay for their transaction to be included in a block. This is known as the transaction gas limit. If the gas usage associated with a transaction surpasses this limit, the EVM raises an Out-of-Gas exception and aborts the associated transaction [22]. Each block has an upper bound on the amount of gas that is determined by the network and the miners. This limit is called the Block Gas Limit. A transaction cannot exceed one block, so the transaction gas limit is also bound by the block gas limit [17]. Therefore, if a transaction requires more gas than Block Gas Limit, it will revert due to Out-of-Gas, which causes the initiation party to waste gas, and no state changes occur. As a result, The execution of one or more functions in a smart contract vulnerable to Out-of-Gas can be blocked indefinitely if Out-of-Gas conditions are not appropriately handled. As a result, DoS attacks can target contracts with gas-related vulnerabilities. One of the principal kinds of gas-based vulnerabilities is *DoS with Block Gas Limit* vulnerability. This vulnerability has a few different types. The most common form that mainly occurs in contracts with user-controlled loops is *DoS with Unbounded Operations*. This can happen when the cost of executing a function exceeds the block gas limit [22]. This can be problematic even without any malicious intent. Generally, loops that user input determines their behavior could iterate too many times, exceeding the Block Gas Limit [13].

```

1 pragma solidity >=0.4.24 <0.7;
2 contract SmallBank{
3     address[] users;
4     function addUsers(address newUser) public {
5         users.push(newUser);
6     }
7     function addInterest(uint interest) public {
8         //Heavy code to compute interest per user
9         for(uint i = 0; i < users.length; i++){
10            users[i].call.value(interest)();
11        }
12    }
13 }

```

Fig. 1 Ethereum smart contract Vulnerable to DoS with Block Gas Limit

Figure 1 demonstrates an example of a contract vulnerable to *DoS with Unbounded Operations*. In this example, the number of iterations in the loop in `addInterest` is determined by the length of the variable `users`, which is controlled by the user input through `addUsers`. `addInterest` performs some expensive arithmetic calculations to compute the interest per user (not shown in the snippet) and then sends each user the interest amount. If the length of `users` is large, the computation required in the loop might reach the block gas limit, which causes the execution of the transaction to reach out of gas and revert. Thus, as the number of `users` grows, the gas needed to execute `addInterest` will increase. Ultimately, the function may become impossible to execute without raising an Out-of-Gas exception, at which point no user can claim their interest, and the `SmallBank` contract will suffer reputation damage and lose users.

3 Description of Gas Gauge

Gas Gauge is designed to address gas-based vulnerabilities of smart contracts. Since loops are the main cause of many gas-related vulnerabilities, the focus of Gas Gauge is on identifying and summarizing loops and then ascertaining whether they are vulnerable. Gas Gauge contains three major parts: the Detection Phase, Identification Phase, and Correction Phase. The Identification Phase and Correction Phase require the information generated by the Detection Phase; however, Identification Phase and Correction Phase are independent of one another. The inputs to all methods are the Solidity source code⁴ and the contract's gas limit (optional if only the Detection Phase is used). Overall, Gas Gauge can detect all the loops and provide a repair to

⁴ The contracts should be self-contained. Thus, contracts with external calls are not supported.

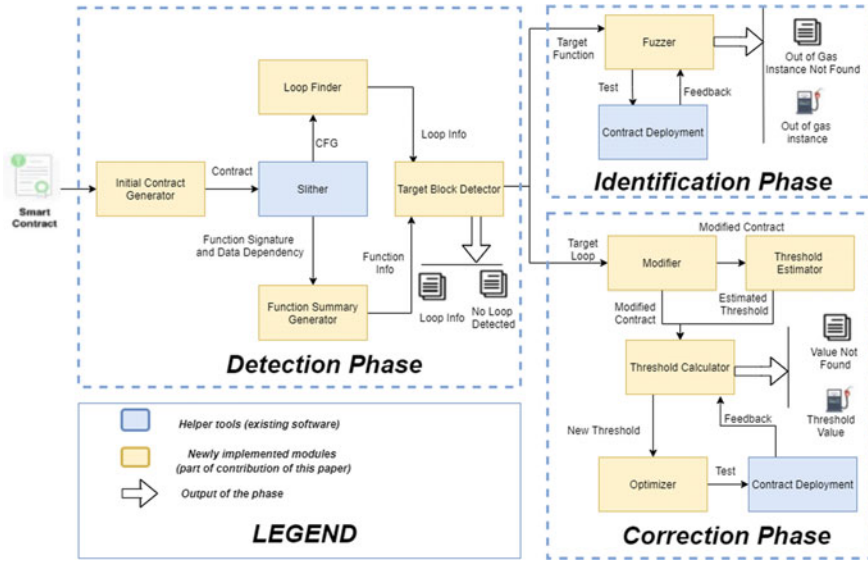


Fig. 2 The architecture of Gas Gauge

contracts vulnerable to gas-related vulnerabilities. The architecture of Gas Gauge is shown in Fig. 2.

3.1 Detection Phase

The Detection Phase uses a static analysis approach to efficiently and accurately detect all the loops in a smart contract.

Initial Contract Generator The first and simplest component of the Detection Phase is the Initial Contract Generator. In this step, a copy of the original contract is made. If the Correction Phase is needed, the copied version is formatted to make it easier for the other parts. For example, it removes all the comments and adds brackets and spaces if needed. Then, the new contract is fed to Slither. This part formats the contract in a way that does not affect the behavior but makes it easier for the static analysis section.

Target Block Detector and Its Inputs In this stage, inputs to the Identification Phase and Correction Phase are generated. The input to the Identification Phase is the target functions, and the input to the Correction Phase is the target loops. First, Slither is used to extract the contract’s Control Flow Graph (CFG) and other useful information like function signatures and data dependencies. The Loop Finder uses the contract’s CFG to find all the loops. Also, the information provided by Slither is used in the Function Summary Generator to summarize the contract. If only the Detection Phase

is needed, the program halts here and outputs the functions containing loops along with the number of loops in each function. If the Identification Phase or Correction Phase is also needed, the types of variables affecting the loop bounds are obtained. This process utilizes a static analysis approach and uses the reports available in Slither to gather the loop conditions, the variables affecting the loops, variable dependencies, and function summaries. The variables affecting the loop bounds are classified into four groups: State, Local, Fixed, and Input variables. State variables are the contract variables whose values are persistently stored in contract storage and can be accessed by all the contract functions. Fixed variables are the ones that only carry a fixed value and are defined within the target function (in the loop bound). Input variables carry their usual meaning: the inputs of the target function. Finally, Local variables are declared and initialized inside the target function, and their context is within a function and cannot be accessed outside. If a Local variable is detected as the loop bound, the Target Block Detector performs induction to find a list of all State, Input, and Fixed variables affecting that local variable. For the white-box fuzzer, the Target Block Detector finds public functions that contain loops with bounds affected by input variables and passes them to the Identification Phase. After identifying the target functions, the function signatures, name, and type of the input variables affecting the target loops are passed to the fuzzer. For the Correction Phase, the function signatures for all the functions containing loops, and a summary of each loop is generated. This summary includes the scope of each loop, the order of the loops if the function contains nested loops, and information about the variables affecting the loop bounds and their types. Therefore, even if a function is not passed to the Identification Phase because it does not satisfy the criteria of this phase, it is still passed to the Correction Phase, and this phase finds the correct threshold values. In the Correction Phase, the contract is slightly modified, so that it can find the thresholds for any loop in any type of function.

3.2 Identification Phase

In the Identification Phase, a white-box fuzzing approach is utilized to generate a set of inputs for each user-controlled loop in a public function in a Solidity smart contract. The bounds of these loops must be affected by at least one of the input variables of the function containing the loop; otherwise, directly fuzzing the target function cannot be effective. This component takes the information from the Detection Phase and the block gas limit as inputs and outputs the set of values that make the transaction go out of gas. Here, public functions mean functions that can be called from outside the contract. Thus, private functions cannot be fuzzed directly without modifying the contract. These functions are supported and checked by the Correction Phase.

In this part, all the input variables in the target function get set to their initial values (i.e., integers are set to zero, and arrays are set to an empty array). The input values reported by the Target Block Detector get encoded in their binary representations. Then the tool picks a bit at random and flips it. Then, the binary encoding gets

converted back to the original form. The only exception is arrays. In this case, the array size can generally affect the bound of a loop, so arrays are represented by 256 bits since arrays in Solidity can have up to 2^{256} elements [5]. Then, the binary representation gets converted to an integer, and the array size gets set to that. If multiple input variables affect the loop bound, the binary representations get concatenated, the bit is flipped, and the concatenated value gets converted back to the original forms. Next, all the necessary files are generated automatically. Truffle Suite [9] is used to deploy the contract to a test Ethereum network. A test file is generated in Solidity to call the target function with generated input values. Then, the contract is deployed, and the target function is fuzzed (tested with different input values). Suppose the test case halts and returns the remaining gas in the contract, the process repeats, and the fuzzer flips another bit. The process continues until a set of inputs is found that makes the test case abort due to an Out-of-Gas exception. At this point, the used set of inputs is reported as the output of the phase.

3.3 Correction Phase

The Identification Phase is convenient to scan the contracts before deployment and check if the contract is at risk of DoS with Block Gas Limit. Since it also provides an instance, it further helps them examine the problem. However, one of the first steps to fixing the code is to find the exact point where the Out-of-Gas condition starts to get triggered, and any arbitrary set of inputs is not enough. Therefore, The Correction Phase is designed to find the upper bound limit of the loops in a smart contract. The output is a formula based on the maximum number of allowed iterations for loops before the transaction runs out of gas. We refer to this number as the threshold of the loop.

Modifier The Modifier makes two copies of the contract generated in the Initial Contract Generator. The first copy is to measure the gas used for each loop's first and second iteration identified by the Target Block Detector. Based on our observation, the first iteration of each loop consumes more gas than the other iterations. This is perhaps due to the gas consumption of the loop initialization, where the counter gets initialized to a starting value. The second iteration's gas usage is typically the average gas usage of all the other loop iterations. The first copy is the input to the Estimator, and the second copy is to change the loop bound to the desired value. It allows us to run each loop with a specified number of iterations and capture the gas left after that many iterations. The Threshold Calculator uses this to test different values. The modifications only have an insignificant effect on the behavior/gas usage of the contract. A public function is added as a wrapper for both modified copies to call the target function.

Threshold Estimator The Threshold Estimator receives the first modified contract and automatically creates a Solidity test file and all the other necessary files for Truffle Suite [9]. Also, to get an actual gas usage for each iteration, the new contract and

Algorithm 1: Estimated threshold for a loop	Algorithm 2: Run-time threshold of a loop in a function
<pre> 1 initial_gas = gasleft() ; 2 iteration = 0 ; 3 while iteration < 2 do 4 original code inside the loop ; 5 if iteration == 0 then 6 gas_1 = initial_gas - gasleft(); 7 end 8 gas_2 = initial_gas - (gas_1 + gasleft()); 9 iteration += 1 ; 10 end 11 max_iteration = 1 + (initial_gas - gas_1)/gas_2 ; </pre>	<pre> 1 guess = estimated value by static and run-time analysis; 2 lower = 0; 3 upper = 5,000; 4 while conditional_expression do 5 gas_left = deploy the contract(guess); 6 if gas_left < 0 then 7 new_gas_left = deploy the contract(guess - 1) ; 8 if new_gas_left < 0 then 9 upper = guess - 1; 10 else 11 threshold = guess - 1; 12 return threshold; 13 end 14 else 15 new_gas_left = deploy the contract(guess + 1) ; 16 if new_gas_left > 0 then 17 lower = guess + 1; 18 else 19 threshold = guess + 1; 20 return threshold; 21 end 22 end 23 guess = upper + lower / 2 ; 24 end </pre>

Fig. 3 The algorithms used in the Correction Phase

test file are deployed to a test Ethereum network using Truffle Suite. The modified contract runs each loop twice and captures the gas used in each iteration. One can call the function `gasleft()` returns (`uint256`) that exists in the global namespace and returns to get the remaining gas at any instance [28]. We can utilize this function to measure the gas usage of the target block of code. Next, based on the gas usage amount reported by the Truffle Suite and the gas limit, the maximum number of iterations that the loop can perform without running out of gas is estimated. Algorithm 1 in Fig. 3 shows the used method. As shown, the maximum iteration is estimated to be the amount of the initial gas before entering the loop minus the gas consumption of the first iteration divided by the average gas consumption of all the other iterations. An extra iteration is added to account for the first iteration.

Threshold Calculator We first use the run-time/static analyzer to estimate the loop bound threshold. Thus, any value over this threshold may trigger the Out-of-Gas condition. Then, we used a run-time verification approach to find a more accurate value. We further use a binary search approach to cut down the action space rapidly. The action space is all the integers that can be the loop bound. The estimated threshold helps the binary search model have a proper starting bound. The second set of the modified files and the Estimator’s result is fed to the Threshold Calculator. The purpose is to run each loop with a specified number of iterations and capture the gas left after that many iterations. The threshold calculator uses this to test different values. Appendix A shows how a contract is modified. At this stage, by inputting different values to the target function, we can calculate different gas usages. The goal here is to find two consecutive numbers, where only the larger number makes the transaction run out of gas. Each loop is isolated to ensure the value is only affecting the target loop. The contract is deployed and run for each value, and the gas left is obtained. There is a lower bound and an upper bound limit for the search space in

the Binary Search model. They are initially set to 0 and 5,000, respectively. Once the execution of a contract passes Truffle's time limit, it throws a time-out exception and stops running. Therefore, if a loop runs for too many iterations, it might trigger the time-out exception. From our experiments, if the maximum number of loop iterations is less than 5,000, the test does not trigger the time-out condition. Also, if a loop threshold is over 5,000, there is a lower chance of running out of gas as most contracts do not require that many iterations in one loop. The algorithm for this part is shown as Algorithm 2 in Fig. 3.

Optimizer The last part is the Optimizer. It has two primary purposes. First, if the lower bound reaches 5,000 or the value reported by the Estimator is over 5,000, it estimates the loop's threshold based on the original gas, gas consumed by the first iteration, gas consumed by 5,000 iterations, and the remaining gas. Secondly, to test a value in the Threshold Calculator, the system needs to run the contract with the target value, wait for the result, and then run the contract with either `value + 1` or `value - 1`. Since each execution takes a few seconds, running the contract twice each time may take a while. Therefore, the Optimizer makes two extra copies of the generated files and simultaneously runs the three contracts with three consecutive numbers (`value`, `value + 1`, `value - 1`). Then, based on the feedback fit receives, it decides to use the correct extra feedback. This way, the run-time gets reduced by almost 50%.

Threshold of the Nested Loops The process for nested loops is slightly different. These loops and their order are identified by the Target Block Detector. Then the Correction Phase finds the threshold for the most inner loop and works its way back to the most outer loop. In order to find the threshold for the inner loops, the loop bounds for the outer loops are set to one, and the threshold for the target loops is found using the method mentioned before. The loop bound for the inner loop is set to zero, and the outer loop threshold is found. The output report contains a formula based on the threshold values of the outer and inner loops. Appendix B demonstrates the mentioned process with a code example.

Output of the Correction Phase The output includes the signature of the functions containing loops and a summary of the loops and the variables affecting the loop bounds. It also provides the value of the threshold found by the tool for the provided gas limit along with the gas consumption of the first iteration and the average gas consumption for the other iterations. Appendix C demonstrates an example of the output of the Correction Phase. Finally, the threshold formula is generated. This formula can be used in a `require` statement to prevent Out-of-Gas exceptions for that loop. The statement has to be placed in the source code right before the loop. The user has to find the loop bound and place it in the `require` statement. An example of the `require` statement is given below.

```
require( loopBound < (gasleft() - gas_1) / (gas_2), "Over the limit")
```

`gasleft()` is the value returned by the function `gasleft()` placed right before the loop in the source code, `gas_1` is the gas consumption of the first iteration

of the loop, and `gas_2` is the average gas consumption of the other iterations. Adding appropriate `require` statements can be a simple fix to many contracts containing user-controlled loops. However, this might not be the case when the contract uses poor implementation patterns. These `require` statements help the callers save some gas (by hitting the `require` statement rather than the block gas limit). Unfortunately, if the contract relies upon some user-controlled loops to function properly, other correction approaches are needed, or the funds are still likely to be locked. However, the information provided by our tool can be useful for the developers to come up with other solutions.

4 Experimental Evaluation

Real-world Smart Contract Benchmark: We gathered a benchmark of 1,000 real-world Solidity smart contracts containing over 60,000 functions from Etherscan.⁵ All contracts are gathered starting from the latest block at the time (Block Height: 11661369) and going backward in the chain. Each contract has 413 lines of code, 63 functions, 4 functions with loops, and 5 loops on average. These contracts were manually checked to ensure that each contained at least one loop. Also, information such as the number of functions, loops, and lines of code was manually collected. Hence, these values are used as the ground truth. Gas Gauge and the benchmark are available at <https://gasgauge.github.io/>. The name of each contract file in the benchmark represents the contract address. The experimental setup can be found in Appendix D.

Competing Tools: The following seven tools were chosen to compare against Gas Gauge: GASTAP [3]/Gasol [2], Madmax [13], MPro [27], Mythril [7], Securify 2.0 [25], Slither [11], and SmartCheck [23]. MPro, Mythril, Securify 2.0, Slither, and SmartCheck were installed using the instruction provided on their official documentation. A combination of GASTAP and Gasol, which is available on a web interface⁶ was used. This interface was called using a script to scan our benchmark. MadMax is built into Contract Library.⁷ Therefore, we searched for our benchmark in Contract Library and collected the reports generated by MadMax. Thus, in our analysis, MadMax and GASTAP/Gasol do not have a run-time value associated with their effectiveness results.

⁵ <https://etherscan.io>.

⁶ <https://costa.fdi.ucm.es/gastap>.

⁷ <https://contract-library.com>.

Table 1 Detection Phase comparison on 1,000 smart contracts with loops

Tool name	Method	Number of contracts Successfully scanned	False negative Rate(%)	Average Run-time (s)
Gas Gauge	Static + Dynamic Analysis	997	0	10
GASTAP/Gasol	Static analysis	120	36	–
Madmax	Static analysis	921	79	–
MPro	Static analysis + Symbolic execution	851	100	242
Mythril	Symbolic execution	870	100	3109
Securify 2.0	Static analysis	548	47	176
Slither	Static analysis	997	85	2.6
SmartCheck	Static analysis	1000	47	2

4.1 Evaluation of the Detection Phase

This experiment aims to determine how accurate and efficient the Gas Gauge and the above-mentioned seven competing tools are at detecting potential loops for Out-of-Gas DoS vulnerabilities. All results are summarized in Table 1. The time-out limit for this experiment was set to 3 hr per contract for each tool. As can be seen, Gas Gauge was able to detect all the vulnerable loops in the contracts that it was able to scan. There were three contracts that Slither could not scan, and since the report generated by Slither is an essential part of our methods, our tool was not successful in checking them as well. Typically, a tool may not scan a contract for various reasons, like reaching the time-out and lacking support for the Pragma version. The false negative rate is calculated as the number of loops detected by the tool divided by the total number of loops in the contracts that the tool could scan. As mentioned before, the number of loops in each contract was manually collected and hence used as the ground truth. Also, we tested all tools on a benchmark of 1,000 contracts without loops, and they all had zero false positive rates. Hence, it is not listed in the table.

One can only obtain the reports of MadMax if a contract exists in Contract Library. Thus, “Number of Contracts Successfully Scanned” shows the number of contracts available on Contract Library. Thus, if a contract does not exist there, it is not necessarily a shortcoming of the tool. However, Madmax was not able to find Out-of-GasDoS vulnerabilities in the Majority of the contracts, although it is one of the industry-standard tools for gas-related vulnerabilities. Also, although GASTAP/Gasol has the lowest false negative rate amongst the competitors, they have a low scan rate of around 12%. The reason is that the Solidity compiler used by GASTAP/Gasol does

not support a majority of the contracts in our benchmark or the generated report did not contain any meaningful information.

Gas Gauge has 0 false positive and false negative rates for the contracts supported by the tool and is efficient in scanning contracts (on average, about 10s per contract). By contrast, all the other tools had difficulties detecting such vulnerabilities. Therefore, Gas Gauge is efficient and effective at detecting loops in smart contracts as it has an average run-time of 10s and false negative and false positive rates of 0%.

4.2 Performance Analysis of the Detection Phase

In this experiment, we tried to find the main factors affecting the run-time of the Detection Phase. We obtain the summary of each of the 1,000 contracts in our benchmark. This summary contains the total number of functions, number of functions containing loops, number of code lines, and number of loops in each contract. We also measured the run-time for each of the contracts for the Detection Phase. During this experiment, we concluded that the factors impacting the run-time of the Detection Phase in descending (run time) order are the number of lines of code, number of loops, number of functions containing loops, and total number of functions. This is also expected because Detection Phase uses a static analysis approach as its primary method. ConsenSys has a tool called Solidity-metrics [8] that has defined a factor called “Complexity Score”, which is a custom complexity score derived from code statements known to introduce code complexity. We obtained this score for each of the contracts and observed that it has a noticeable correlation with the run-time and can summarize all the other factors. Appendix E contains graphs that show the impact of different factors on the run-time of the Detection Phase.

This experiment cannot be done fairly on the other two phases since they contain run-time analysis. Thus, the time to deploy and run the contracts has a significant impact on the overall run-time of the tool. Furthermore, for the Identification Phase, factors like the search space, the ability of the fuzzer to find the right set of inputs, and the complexity of the target functions. Also, for the Correction Phase, the estimated value reported by the Threshold Estimator, cyclomatic complexity of the loop, and the actual threshold values are some of the factors that affect the run-time noticeably and cannot be measured easily.

4.3 Evaluation of the Identification Phase

In this experiment, we evaluated the results of the Identification Phase. Generally, finding an Out-of-Gas instance requires a run-time analysis-based technique. To the best of our knowledge, Gas Gauge is the first tool, and the Identification Phase is the first method that uses a run-time fuzzing technique to detect gas-related vulnerabilities and automatically identify Out-of-Gas instances. As a result, we did not

find a direct competitor to compare the results of the Identification Phase. Therefore, we manually checked each contract to obtain the number of functions satisfying the condition of the fuzzer and the input variables affecting the loops. Only 979 functions in 501 contracts met these criteria. Then, we ran our tool on the benchmark and obtained the results. Lastly, we verified the results manually. The fuzzer detected 968 functions in 499 contracts and identified their variables correctly in 53 s per contract on average. Two of the contracts were not scanned by Slither, so the fuzzer was not able to identify 11 functions. The fuzzer identified 614 instances of Out-of-Gas in 331 contracts. Although the fuzzer detected the rest, it could not find an Out-of-Gas instance in them for various reasons, like reaching the maximum number of tries (it was set to 10, but it is customizable), the function contained structs, or the code reverted. However, even when the fuzzer could not find an out of the gas instance, it still provided the function signatures with loops and variables affecting the loop bounds. Moreover, different approaches that were considered when designing the fuzzer are described in Appendix F. Overall, the Identification Phase identified all the satisfying functions that it could scan. It also was able to identify 614 instances of Out-of-Gas in 331 contracts.

4.4 The Evaluation of the Correction Phase

In this experiment, we evaluated the results of the Correction Phase. Our benchmark has 4415 loops. Gas Gauge was able to find the thresholds for 932 loops in 467 contracts. 779 of these thresholds were calculated using run-time verification with a high rate of accuracy, and 153 of these thresholds were estimated with 95% accuracy. A threshold is estimated if it is greater than 5,000 or for any reason, the run-time verification is not able to find the correct number. The average run-time for each loop was about 389 s, which is much faster than manual processing. To the best of our knowledge, Gas Gauge is the first tool ever attempted to find the loop upper bound limits in a smart contract. The closest tool to our work is GASTAP/Gasol. However, based on our experiments, the provided web interface does not support most of the contracts in our benchmark. Meanwhile, manually finding the thresholds is a tedious process. Therefore, to get an idea of the accuracy, we randomly picked 10 of the contracts and found the thresholds of their loops manually. Then, we compared the manual results with the ones generated by Gas Gauge. Based on our results, the calculated threshold is about 2% lower than the actual threshold. We expect the calculated thresholds to be within 5% of the actual values and usually lower than the actual values since our modifications introduce an insignificant extra gas usage.

4.5 Limitations of Gas Gauge

If the Slither tool is not able to scan a contract or does not identify loops or data dependencies, then all three phases of Gas Gauge miss important information that lowers its accuracy. Also, the time limit and compilation problems of Truffle Suite may cause our tool not to operate properly. Functions containing structs or multi-dimensional arrays are not supported by the Correction Phase and Identification Phase. A struct is just a custom type that can be defined within a contract, and because it is different in each contract, it can be very challenging to automate the test generation for such a construct. However, they are supported in the Detection Phase, which is the phase that competes with other tools. Moreover, the contracts should be self-contained and written in Solidity to be used by Gas Gauge. Thus, contracts with external calls to other contracts or written in other languages are not supported.

5 Case Study

We performed a case study to evaluate Gas Gauge in real-world applications. Quantstamp [21] is a leading verification company that evaluates smart contract projects for security-related issues and code quality. We collected the Quantstamp contract security certification of Airswap,⁸ a peer-to-peer trading network for Ethereum, to identify gas-related vulnerabilities in this project. Airswap is used for trading the USD equivalent of about 9200 ETH/week (over 20 million USD/week with current ETH value). Loop concerns due to gas usage have been reported in two contracts, `Swap.sol`, and `Index.sol`. Gas Gauge was able to identify one vulnerable function in `Swap.sol` and two vulnerable functions in `Index.sol`. Gas Gauge detected the loop and variables affecting the loop bound in `Swap.sol`. Although it did not find the threshold of the loop since the constructor of that contract takes a struct as an input, it detected the loops and variables affecting the loop bounds of `Index.sol`. For the two other functions, it found the threshold of the loop as well. The threshold of one of these loops was over 5,000, so Gas Gauge estimated the threshold based on the average gas consumption of a few iterations.

Usually, to find the loop threshold values, one needs to audit the contracts to find potentially vulnerable code blocks. The next step is to examine these code blocks to understand what variables affect the loop bounds. Once all the information is gathered, a gas analysis needs to be performed. The gas analysis consists of making required files for tools like Truffle Suite to test a contract with different test cases to find the threshold for each loop.⁹ This process is tedious and requires many man-hours (estimated to be around 4 hr per contract) of work by the developers. However, Gas Gauge took about 10m to find the vulnerable functions and loop thresholds. Also, Gas Gauge is automatic and requires limited resources and supervision. This

⁸ Available at <https://certificate.quantstamp.com/full/airswap>.

⁹ The process can be found at [here](#) and [here](#).

case study shows that Gas Gauge is practical and can be used to save hours of manual work.

6 Related Work

Many smart contract verification tools scan a contract for multiple security vulnerabilities. Some of the most well-known tools are Manticore [18], MPro [27], Mythril [7], Securify [24] (deprecated), Securify 2.0 [25], Slither [11], SmartCheck [23], Verx [20], and Zeus [16]. Although these tools can detect multiple security vulnerabilities, most of them either cannot identify gas-related vulnerabilities or their results are not reliable due to their high false negative rate (See Table 1). Fuzzing tools like ContractFuzzer [15], Echidna [14], and Harvey [26] do not discover gas-related vulnerabilities, to the best of our knowledge. Meanwhile, some research has focused on gas-related vulnerabilities. GASTAP [3] derives gas upper bounds for all public functions of smart contracts via inferring size relations, generating gas equations, and solving these equations. Madmax [13] uses a static program analysis technique to detect gas-focused vulnerabilities automatically. However, most of these tools cannot detect gas-related DoS vulnerabilities directly, or they do not provide any information to fix the problem.

Gas Gauge can find all the loops in a contract reliably and quickly. Also, it identifies the exact functions and variables and provides an Out-of-Gas instance that helps developers investigate the problem further. Finally, to the best of our knowledge, Gas Gauge is the only tool that accurately and reliably finds the threshold values and provides more useful information like each loop's type and the variables affecting it. This information is helpful in order to repairing the code and preventing gas-based attacks like DoS with Block Gas Limit.

7 Conclusions and Future Work

Because smart contracts contain monetary transactions, it is crucial to make sure they are risk-free. This paper summarizes the design and implementation of Gas Gauge, an automatic tool that helps developers and contract owners identify DoS with Block Gas Limit vulnerability and repair their code. This tool contains three powerful sections. These sections use static analysis, run-time verification and white-box fuzzing to detect all the contract loops, provide an instance of out-of-gas and determine when the transaction starts to go out of gas. Finally, our experimental evaluation results on 1,000 real-world Solidity smart contracts show that all the methods are accurate and efficient. Gas Gauge only supports self-contained contracts, so contracts with external calls to other contracts are not supported. As a part of future work, the Mainnet forking will be used to include external contracts without source code. Also, The current implementation only supports contracts written in Solidity. As an

improvement, we plan to extend our tool so that it can support more programming languages like Vyper and Rust. Furthermore, the fuzzer can be improved so it can identify the private functions containing loops and fuzz the public functions that call those private ones.

Appendix A Threshold Calculator Code Modification

The code snippets in Figs. 4 and 5 demonstrate how a contract is modified in order to find the threshold for the first loop. The code snippet in Fig. 4 is the original contract that contains two loops and the code snippet in Fig. 5 is the modified contract.

As shown, a wrapper function is added, the visibility of the target function is changed from “external” to “public”, the second loop’s condition is modified, so it does not get executed, and the condition of the target loop is changed to execute the loop up to a desired number of iterations. `VALUE` is the number generated by the Binary Search Threshold Calculator. Also, if the original loop bound is affected by an input variable as identified in Target Block Detector, the generated test file is modified, so the value or size of the input variable in the wrapper function is set to `VALUE` where applicable. This ensures the function does not fail if there is a condition that requires a specific value or size of that variable inside the function. For example, in the shown code snippet, `input1` is set to an array of integer of size `VALUE`. Similarly, if a state variable affects the loop bound, the value or size of the variable is set to `VALUE` inside the wrapper function before calling the target function. For example, to test the second loop in the given code snippet, “numbers” is set to an array of integers of size `VALUE` before calling `addNumbers`.

```
1 pragma solidity >=0.4.24 <0.7;
2 contract TestContract{
3     uint[] numbers;
4     function addNumbers(uint[] calldata newNumbers) external returns(
5         uint){
6         for(uint i=0; i<newNumbers.length;i++) {
7             numbers.push(newNumbers[i]);
8         }
9         uint sum = 0;
10        for(uint j=0; j<numbers.length; j++){
11            sum += numbers[j];
12        }
13 }
```

Fig. 4 The original code containing two loops

```

1 pragma solidity >=0.4.24 <0.7;
2 contract TestContract {
3     uint[] numbers;
4     function getStateVarInaddNumbers(uint[] memory input1) public
      returns (uint) {
5         addNumbers(input1);
6         return gasleft();
7     }
8     function addNumbers(uint[] memory newNumbers) public returns (uint)
      {
9         uint loopCounter = 0;
10        for(uint i=0; loopCounter < VALUE; i++) {
11            numbers.push(newNumbers[i]);
12            loopCounter = loopCounter + 1;
13        }
14        uint sum = 0;
15        for(uint j=0; 1 == 0 ; j++){
16            sum += numbers[j];
17        }
18    }
19 }

```

Fig. 5 The modified code containing two loops

```

1 function hasCardExpired(uint[] calldata marketAddresses, uint
  numberOfTokens) public returns (bool) {
2     bool _expired = false;
3     for (uint i = 0; i < marketAddresses.length; i++) {
4         IRealityCards rc = IRealityCards(marketAddresses[i]);
5         for (uint j = 0; j < numberOfTokens; j++) {
6             if (rc.cOwnerLeftDeposit(j) == 0 && rc.ownerOf(j) != address(rc
7             )) {
8                 _expired = true;
9             }
10        }
11    }
12    return _expired;

```

Fig. 6 The original code containing a nested loop

Appendix B Code Modification for Nested Loops

The code snippets in Figs. 6, 7 and 8 demonstrate how a contract with nested loops are modified. The first code snippet shows the original function containing a nested loop. The second and third code snippets show the changes in the code in order to obtain the threshold for the inner and outer loops respectively.

```

1  function hasCardExpired(uint[] calldata marketAddresses, uint
    numberOfTokens) public returns (bool) {
2  bool _expired = false;
3  uint outerLoopCounter = 0;
4  for (uint i = 0; outerLoopCounter < 1; i++) {
5      IRealityCards rc = IRealityCards(marketAddresses[i]);
6      uint loopCounter = 0;
7      for (uint j = 0; loopCounter < VALUE; j++) {
8          if (rc.cOwnerLeftDeposit(j) == 0 && rc.ownerOf(j) != address(rc
    )) {
9              _expired = true;
10             }
11             loopCounter = loopCounter + 1;
12         }
13         outerLoopCounter = outerLoopCounter + 1;
14     }
15     return _expired;
16 }

```

Fig. 7 The modified code for the threshold of the inner loop

```

1  function hasCardExpired(uint[] calldata marketAddresses, uint
    numberOfTokens) public returns (bool) {
2  bool _expired = false;
3  uint outerLoopCounter = 0;
4  for (uint i = 0; outerLoopCounter < VALUE; i++) {
5      IRealityCards rc = IRealityCards(marketAddresses[i]);
6      for (uint j = 0; 1 == 0; j++) {
7          if (rc.cOwnerLeftDeposit(j) == 0 && rc.ownerOf(j) != address(rc
    )) {
8              _expired = true;
9              }
10         }
11         outerLoopCounter = outerLoopCounter + 1;
12     }
13     return _expired;
14 }

```

Fig. 8 The modified code for the threshold of the outer loop

Appendix C Output of the Correction Phase

The threshold formula of the Correction Phase is given in the following format:

$$(gasleft() - gas_1)/(gas_2).$$

If there are nested loops in the contract, the formula is slightly different. The formula for the inner loops is similar to the ones above, but it considers the outer loop has only one iteration, and for the outer loops is similar to the following:

$$(gasleft() - gas_1)/(gas_2 + Internal).$$

```

1 Report for 0xd848f9b61affaaa2e5a7402e87d27eaa0cc27b6f.sol:
2 Found 1 loop in 1 function with the following information:
3 Function Name: VAPE.multiTransfer(address[],uint256[]), Visibility: public, Number of Loops: 1
4 -----
5 Results for VAPE.multiTransfer(address[],uint256[]):
6 inputs affecting the function loop bound(s) are: input 1 (address[] memory receivers);
7
8 Results of the Threshold Finder:
9 for loop 1 in VAPE.multiTransfer(address[],uint256[]):
10 The type is Normal
11 For gas limit = 6721975, the calculated threshold = 899
12 gas consumption for the first iteration = 47420; average gas consumption for other iterations = 5932
13 Threshold formula (gasleft() is the global function gasleft() placed before entering the loop ) =
14 (gasleft() - 47420) / (5932)
15 -----
16 Run time = 422.0240927560001 seconds

```

Fig. 9 Output of Correction Phase

Where `gasleft()` is the value returned by the function `gasleft()` placed right before the loop in the source code, `gas_1` is the gas consumption of the first iteration of the loop, `gas_2` is the average gas consumption of the other iterations, and `Internal` is the gas consumption of the internal loops.

An example of the output of the Correction Phase is provided in Fig. 9.

In this example, the size of the input variable “receivers” is the bound of the loop, the “require” statement looks like this:

```

require(receivers.length < (gasleft() - 47420)/(5932),
“Loop bound is over the threshold!”);

```

In this case, if the user inputs an array of size greater than 899, the “require” statement gets triggered, and the execution stops before entering the loop.

Appendix D Experimental Setup

We ran our experiments on a machine that was equipped with 8GB of RAM, a 4-core Intel Xeon 2.2 GHz processor with Ubuntu 18.04 running. To test Gas Gauge, the latest version (on January 2021) of nodejs, Ethereum, truffle, ganache-cli, solc-select, python3, and Slither were installed. We deployed the target smart contract to a test chain using Ganache-cli. Since Gas Gauge does not make any modifications to Truffle and Ganache-cli in its implementation, future versions of these tools are still compatible with Gas Gauge. In all these experiments, Solc compiler version 0.5.3 was used unless either the tool or the contract required a different version. Also, the block gas limit was so kept to the default value in Ganache-cli (6,721,975). The Solc compiler version and the block gas limit can be configured easily in the tool.

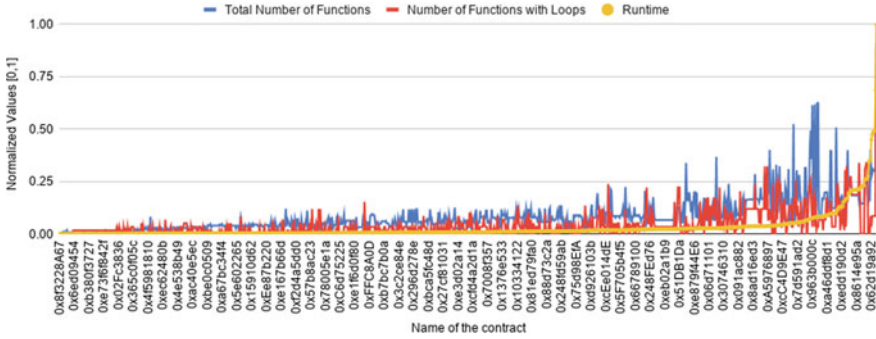


Fig. 10 Effect of the number of functions on the run-time

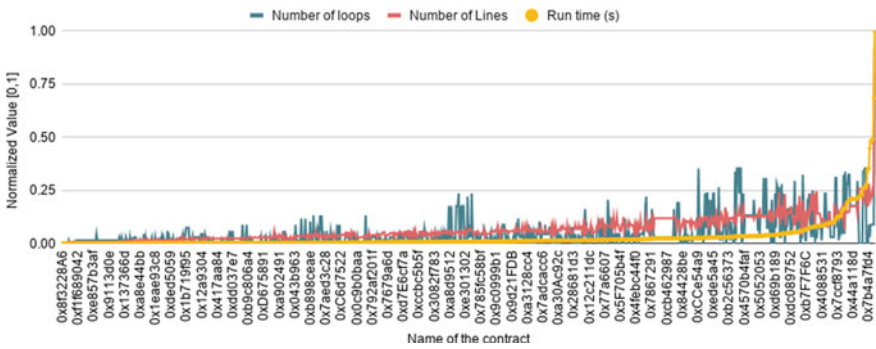


Fig. 11 Effect of the number of loops and line of code on the run-time

Appendix E Factors Impacting the Run-time of the Detection Phase

Figures 10 and 11 demonstrate the impact of different factors on the run-time of the Detection Phase. These factors are the number of lines of code, number of loops, number of functions containing loops, and total number of functions. These factors and some other factors can be summarized in the “Complexity Score”, provided by ConsenSys. Figure 12 demonstrates the impact of this factor on the run-time of Detection Phase.

Appendix F Evaluation of the Methods for the Identification Phase

Three methods were considered when designing the white-box fuzzer. The first approach was a random bit flip, as described before. The second one was a ran-

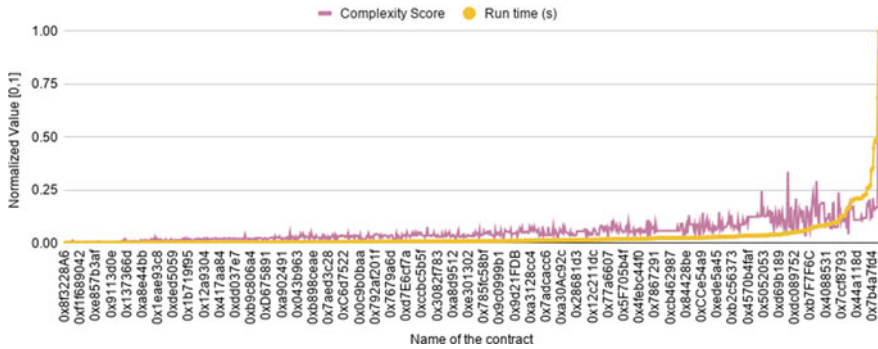


Fig. 12 Effect of the complexity score on the run-time

dom byte flip, which flips every bit in the byte starting from the randomly chosen bit. Lastly, a random byte shuffle was tested. In this approach, the fuzzer chooses a random bit to flip, and then all the bits in the byte starting from the chosen bit get randomly shuffled. Figure 13 shows a chart comparing the three methods on a benchmark of 28 contracts containing a total of 31 functions with loops. We tested each method ten times on each function and recorded the average number of tries for each method until finding a set of inputs that causes the Out-of-Gas exception. The number of tries means the number of different combinations of inputs before finding a satisfying set. This number was bounded to fifty in our experiments in order to halt the process promptly. During this experiment, we obtained the following results:

1. Because our implementation finds the exact functions and variables to fuzz, in most cases, all three methods can find the desired output within the first two tries. These are simple cases when one input variable is the loop bound, so the fuzzer has a high chance of finding the correct value for the value.
2. In some cases, when the bound is more complex, the fuzzer takes about 3 or 4 tries to find a correct set of inputs using any of the approaches. An example of this is when the difference between the values of two of the inputs is the loop bound.
3. There are also some cases that the fuzzer needs to try more numbers. An example of this is when “input mod 250” is the loop bound, and any number of loop iterations more than 240 triggers the Out-of-Gas condition. These contracts were the determining factors, and as shown, bit flip outperformed the other two methods. Hence, the bit flip approach was chosen in our design.

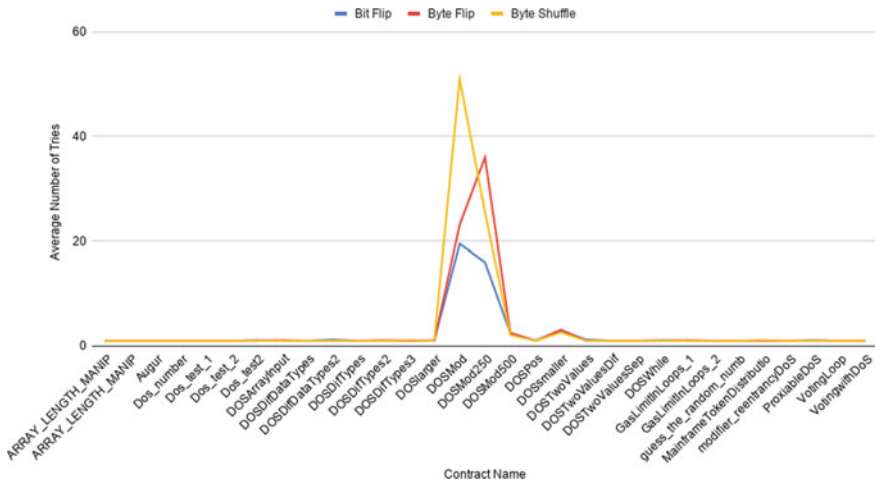


Fig. 13 A comparison of different methods for the white-box fuzzer

References

1. If metamask gas calculations are nearly perfect why do we still get out of gas error? (2018). <https://ethereum.stackexchange.com/questions/56287>.
2. Albert, E., Correias, J., Gordillo, P., Román-Díez, G., & Rubio, A. (2019). Gasol: Gas analysis and optimization for ethereum smart contracts.
3. Albert, E., Gordillo, P., Rubio, A., & Sergey, I. (2019). Running on fumes—preventing out-of-gas vulnerabilities in ethereum smart contracts using static resource analysis. <https://arxiv.org/abs/1811.10403>.
4. Calderon, F. (2021). Why did my transaction fail with an out of gas error? how can i fix it?. <https://metamask.zendesk.com/hc/en-us/articles/360038849792/>.
5. Cañada, A. C. (2019). How not to run out of gas in ethereum. <https://hackernoon.com/how-much-can-i-do-in-a-block-163q3xp2>.
6. ConsenSys Software Inc. (2021). Metamask. <https://metamask.io/>.
7. ConsenSys Software Inc. (2021). Mythril. <https://github.com/ConsenSys/mythril>.
8. ConsenSys Software Inc. (2021). solidity-metrics. <https://github.com/ConsenSys/solidity-metrics>.
9. ConsenSys Software Inc. (2021). Sweet tools for smart contracts. <https://www.trufflesuite.com/>.
10. Ethereum. (2021). Solidity. <https://docs.soliditylang.org/>.
11. Feist, J., Grieco, G., & Groce, A. (2019). Slither: A static analysis framework for smart contracts. In *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*. <https://doi.org/10.1109/wetseb.2019.00008>, <https://arxiv.org/abs/1908.09878>.
12. Godefroid, L. M. (2012). Sage: Whitebox fuzzing for security testing. <https://queue.acm.org/detail.cfm?id=2094081>.
13. Grech, N., Kong, M., Jurisevic, A., Brent, L., Scholz, B., & Smaragdakis, Y. (2018). Mad-max: surviving out-of-gas conditions in ethereum smart contracts. In *Proceedings of the ACM on Programming Languages* (vol. 2, pp. 1–27). OOPSLA. <https://doi.org/10.1145/3276486>, <https://dl.acm.org/doi/10.1145/3276486>.

14. Grieco, G., Song, W., Cygan, A., Feist, J., & Groce, A. (2020). Echidna: effective, usable, and fast fuzzing for smart contracts. In *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*. <https://doi.org/10.1145/3395363.3404366>.
15. Jiang, B., Liu, Y., & Chan, W. K. (2018). Contractfuzzer: fuzzing smart contracts for vulnerability detection. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. <https://doi.org/10.1145/3238147.3238177>, <http://dx.doi.org/10.1145/3238147.3238177>.
16. Kalra, S., Goel, S., Dhawan, M., & Sharma, S. (2018). Zeus: Analyzing safety of smart contracts. In *Proceedings 2018 Network and Distributed System Security Symposium*. <https://doi.org/10.14722/ndss.2018.23082>.
17. minimalism: Gas and fees. <https://ethereum.org/en/developers/docs/gas/>.
18. Mossberg, M., Manzano, F., Hennenfent, E., Groce, A., Grieco, G., Feist, J., Brunson, T., & Dinaburg, A. (2019). Manticore: A user-friendly symbolic execution framework for binaries and smart contracts. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 1186–1189). California: IEEE. <https://doi.org/10.1109/ASE.2019.00133>.
19. OpenZeppelin: Deploying smart contracts using create2 (2018). <https://docs.openzeppelin.com/cli/2.8/deploying-with-create2>.
20. Permenev, A., Dimitrov, D., Tsankov, P., Drachler-Cohen, D., & Vechev, M. (2020). Verx: Safety verification of smart contracts. In *2020 IEEE Symposium on Security and Privacy (SP)* (pp. 1661–1677). California: IEEE. <https://doi.org/10.1109/SP40000.2020.00024>.
21. Quantstamp Inc. Quantstamp certifications. <https://certificate.quantstamp.com>.
22. SmartContractSecurity: Swc registry smart contract weakness classification and test cases. <https://swcregistry.io/docs/SWC-128>.
23. Tikhomirov, S., Voskresenskaya, E., Ivanitskiy, I., Takhaviev, R., Marchenko, E., & Alexandrov, Y. (2018). Smartcheck: Static analysis of ethereum smart contracts. In *2018 IEEE/ACM 1st International Workshop on Emerging Trends in Software Engineering for Blockchain (WET-SEB)* (pp. 9–16). Gothenburg: IEEE. <https://doi.org/10.1145/3194113.3194115>.
24. Tsankov, P., Dan, A., Drachler-Cohen, D., Gervais, A., Bünzli, F., & Vechev, M. (2018). Securify: Practical security analysis of smart contracts. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. <https://doi.org/10.1145/3243734.3243780>, <https://dl.acm.org/doi/10.1145/3243734.3243780>.
25. Tsankov, P., Dan, A., Drachler-Cohen, D., Gervais, A., Bünzli, F., & Vechev, M. (2021). Securify v2.0. <https://github.com/eth-sri/securify2>.
26. Wüstholtz, V., & Christakis, M. (2020). Harvey: a greybox fuzzer for smart contracts. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. <https://doi.org/10.1145/3368089.3417064>.
27. Zhang, W., Banescu, S., Pasos, L., Stewart, S., & Ganesh, V. (2019). Mpro: Combining static and symbolic analysis for scalable testing of smart contract. In *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)* (pp. 456–462). Berlin: IEEE. <https://doi.org/10.1109/ISSRE.2019.00052>.
28. Ziechmann, K. (2021). Introduction to smart contracts. <https://ethereum.org/en/developers/docs/smart-contracts/>.

Tweakable $\mathcal{S}_{\text{leeve}}$: A Novel $\mathcal{S}_{\text{leeve}}$ Construction Based on Tweakable Hash Functions



David Chaum, Mario Larangeira, and Mario Yaksetig

Abstract Recently, Chaum et al. (ACNS'21) introduced $\mathcal{S}_{\text{leeve}}$, which describes an extra security layer for signature schemes, *i.e.*, ECDSA. This distinctive feature is a new key generation mechanism, allowing users to generate a “back up key” securely nested inside the secret key of a signature scheme. Using this novel construction, the “back up key”, which is secret, can be used to generate a “proof of ownership”, *i.e.*, only the rightful owner of this secret key can generate such a proof. This design offers a quantum secure *fallback*, *i.e.*, a brand new quantum resistant signature, ready to be used, nested in the ECDSA secret key. In this work, we rely on the original $\mathcal{S}_{\text{leeve}}$ definition to generalize the construction to a modular design based on *Tweakable Hash Functions*, thus yielding a cleaner design of the primitive. Furthermore, we provide a thorough security analysis taking into account the security of the ECDSA signature scheme, which is lacking in the original work. Finally, we provide an analysis based on formal methods using *Verifpal* assuring the security guarantees our construction provides.

Keywords Provable security · Digital wallet · Hash-based signatures

This work was supported by JSPS KAKENHI Grant Number JP21K11882.

D. Chaum
xx Network, Grand Cayman, Cayman Islands
e-mail: david@xx.network

M. Larangeira (✉)
Tokyo Institute of Technology, Tokyo, Japan
e-mail: mario@c.titech.ac.jp; mario.larangeira@iohk.io

IOHK, Battery Road, Singapore

M. Yaksetig
University of Porto, Porto, Portugal
e-mail: mario.yaksetig@fe.up.pt

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes
in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_10

1 Introduction

The ECDSA signature scheme is widely used; however it achieved new levels of exposure after it found new applications in electronic wallets for cryptocurrencies such as Bitcoin [22], Ethereum [25] and Cardano/Ouroboros [2, 11, 19]. This intensive exposure drove the research community to channel its efforts to propose new attacks on the signature scheme/wallets [1, 24].

Recently, Chaum et al. [8] proposed $\mathcal{S}_{\text{leeve}}$, a signature based new cryptographic primitive in order to mitigate damages during massive leaks of wallet private information. In a nutshell, the goal of [8] is to allow the rightful user to prove its (correct) ownership in the face of the situation that its secret key becomes public. In such a situation, proving the knowledge of the correct secret key, via zero knowledge protocols, for example, is of no use as, potentially, anyone could generate such proof. Furthermore, $\mathcal{S}_{\text{leeve}}$ leaves at the disposal of the user a second signature scheme. More concretely, $\mathcal{S}_{\text{leeve}}$ leverages a regular ECDSA scheme to have a nested “back up key” to generate the proof of ownership, or even be fully discarded for a (post quantum) signature scheme; a hash based signature scheme. In theory, wallets implementing $\mathcal{S}_{\text{leeve}}$ can be easily switched to be quantum resistant, since in addition to ECDSA, they would also contain a post quantum signature as the fallback feature.

$\mathcal{S}_{\text{leeve}}$ Design Limitations. The novel approach in [8], in particular the construction the original authors introduced, deals with the aggregation of a W-OTS^+ public key into a single value to be used in the ECDSA as the secret key. Their solution was to adapt the L-Tree data structure [10] in order to execute the integration. This approach works for their purpose; however its design seems fairly limited and ad hoc, *i.e.* left and right branches of the L-Tree requires pair of values which needs to be added to the key pairs. More modern approaches exist and seem more suitable to such integration between ECDSA and hash based signatures, such as relying on Tweakable Hash Functions [3]. The security analysis of [8] introduces two new properties: *proof of ownership* and *fallback*; however it does not detail the impacts in the signature scheme. Namely, the introduction of a back up key nested into the ECDSA signature scheme is not shown to have any side effects on its security. In fact, the ECDSA security in [8] is not assured by a computational problem. Transactions generated by wallets rely on signatures, therefore this state of uncertainty is not ideal for the security of regular users. Moreover, a closer look on the ECDSA security literature shows that it is more involved than a naive reader would previously expect [6, 7, 13, 14].

History of the ECDSA Security. Brown [6, 7] has shown that the ECDSA is strong unforgeable (when the adversary receives only one signature per message) in a chosen message attack considering a proof technique based on the Random Oracle Model (ROM) and Generic Group Model (GGM). Fersch et al. [13] pointed out, that indeed ECDSA is strong unforgeable in these models; however in the real world, when no assumption is assumed in the group (thus not in the GGM), that is not the case. The

reason for the discrepancy is the modelling of the group in the conversion function of the scheme, *i.e.*, mapping the group elements to the field \mathbb{Z}_q for a large prime q .

The works [13, 14] sidestep the briefly mentioned limitations of the proof technique by dropping the GGM, while still relying on the ROM. Both works show that ECDSA is indeed secure; however when the adversary is given only one signature per message employing a proof method relying on a “Generic ElGamal Framework”, which subsumes several variants of DSA, including the ECDSA. Perhaps, surprisingly, the proven security is based in the Semi Logarithm Problem (SLP) [6] instead of the more standard Discrete Logarithm Problem (DLP) as one would expect. Attempts to show the security of $\mathcal{S}_{\text{leeve}}$ must take into account these developments, and that is what we do within this work.

Our Contributions. Succinctly, the main contributions of this work are

- Section 3 (and Appendix A) introduces a clean and modular construction to quantum-secure fallback and proof of ownership of ECDSA under the $\mathcal{S}_{\text{leeve}}$ definition and based on Tweakable Hash Functions;
- Section 4 presents a proof of security with respect to the original $\mathcal{S}_{\text{leeve}}$ definition, for proof of ownership and fallback, regarding the signatures generated by $\mathcal{S}_{\text{leeve}}$ with respect to the unforgeability security of the ECDSA scheme, and based on the computational problem SLP (dependence of a computational problem is crucial in provable security standard);
- Section 5 introduces benchmarks of an open-source, fully audited, and deployed implementation currently in use on existing blockchain platform;
- Section 6 shows the security of the construction using Verifpal, a formal methods analysis tool, and provides the first ever analysis of a hash-based signature scheme using formal methods analysis tools, highlighting the existing challenges in this type of modelling.

The most remarkable differences between the work from [8] and ours is (1) the use of Tweakable Hash functions, which [8] does not use. Therefore, as in [8], our construction works with basic wallet scripts, *e.g.*, multisig. Their construction relies heavily on L-Tree as used in [16], therefore our construction takes the more modern approach, (2) the security guarantees and analysis we introduce.

2 Background

As preparation, let n be the security parameter, and PPT denote *probabilistic polynomial-time*. We rely on the standard notion of negligible function. That is $\text{negl}(n)$ is said to be negligible if and only if for all $c \in \mathbb{N}$, there is a n_0 such that for all $n \geq n_0$, $\text{negl}(n) < n^{-c}$.

Now we can review the $\mathcal{S}_{\text{leeve}}$ and Tweakable Hash Function definitions.

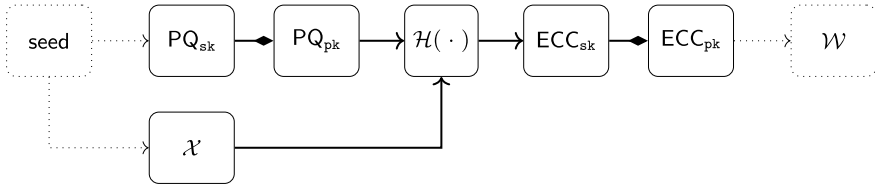


Fig. 1 Overview of the \mathcal{S}_{leeve} construction, where the user generates a post-quantum (PQ) key pair (PQ_{sk} , PQ_{pk}) along with a hash key \mathcal{X} from the local randomness **seed**, which is used as an input when hashing the fallback public key. The result of the hashing operation is used as an elliptic curve secret key, ECC_{sk} , then used as elliptic curve cryptography (ECC) trapdoor and obtain the elliptic curve public key ECC_{pk} . Diamond arrows represent a trapdoor, and normal arrows showcase the values acting either as input or output

Overview of the \mathcal{S}_{leeve} construction. The principle behind the construction is that users first generate a public-private key pair that is quantum resistant along with a secret key value, hash the quantum-resistant public key along with the secret key value, and use this output as a secret key to be used as an elliptic curve secret key. Upon obtaining the elliptic curve secret key, users can trivially generate the corresponding public key. To finalize, users may have to perform additional steps to obtain a wallet address associated with an elliptic curve public key. For example, on blockchain platforms like Bitcoin [22], Ethereum [25] and Cardano/Ouroboros [2, 11, 19], users hash their ECDSA public key to obtain their wallet address. The construction is designed to be modular such that users can easily use the best suitable cryptographic assumptions for each of the modules. Figure 1 illustrates an overview diagram of the construction.

\mathcal{S}_{leeve} Desired Properties. The design [8] is due to the need to integrate a quantum-secure fallback into the ECDSA scheme. Namely, the question it addresses is: *If an adversary breaks the Elliptic Curve based DLP (ECDLP), compromising the security of a cryptocurrency, can users redeem (or rollover) their assets in a safe manner without the risk of theft from this adversary?*

Before addressing the required properties for our design, we highlight similar research in this area, such as [18] and [17], that provide a different alternative solution to this question. These approaches rely on a user Alice publishing one hash commit to both the elliptic curve public key and the fallback public key. At a later point in time, Alice signs a reveal transaction using the fallback secret key which reveals both the elliptic curve public key and the fallback key. This transaction then proves that Alice is the true owner of a specific wallet address.

The scheme in [8] requires some different properties, which are now enumerated. First, and intuitively, users should have the ability to integrate a (quantum-secure) fallback for traditional cryptocurrency wallets, which typically rely on the ECDSA scheme. Ideally, this solution should not incur in any type of additional communication costs and should not assume an interactive protocol if it is not strictly necessary. This segregation and lack of interactivity with any other parties is particularly rele-

vant as it allows a user to, upon completion of the key generation, quickly and simply store the fallback key in a cold wallet without requiring any signature until a quantum threat appears. Second, the users should have the ability to leak the fallback public key in a manner that does not expose the ECDSA secret key. For example, Alice should be able to disclose that she owns a wallet address \mathcal{W}_A , and a fallback public key to inform all in the system that she may need to provide a signature that can be verified under such fallback key. The reveal of the fallback public key should not translate to a compromise of the ECDSA secret key as then any entity in the system could produce signatures and attempt to perform transactions on behalf of Alice.

Third, the design should be modular, easy to use, and compatible with currently used cryptocurrency wallets. Therefore, the design should have the possibility of supporting any elliptic curve based wallet, any post-quantum secure fallback, and should support the use of mnemonics and other features that improve usability for the end user. Ideally, the security proofs for each of the components should be modular such that changing the used schemes in the different parts of the design does not affect other parts of the construction.

Lastly, one of the main properties of this construction is *fork voiding* in a blockchain system. Upon redeeming the digital assets into the fallback public key, users should fully expose the ECDSA secret key such that the value of the assets stored on the original chain naturally converges towards zero, thus it incentives users to abandon the initial chain towards the new and safer fork.

$\mathcal{S}_{\text{leeve}}$ and its Security Properties. The $\mathcal{S}_{\text{leeve}}$ primitive is composed by the tuple $(\text{Gen}_\pi, \text{Sign}, \text{Verify}, \text{Proof}, \text{Verify-Proof})$. The generation algorithm outputs the pairs of keys, $\forall k$ and $s k$, and the backup key $b k$. The first pair is the regular verification key, used for verifying a signature, and the secret-key used for issuing a signature. While the last key is used to issue the *Proof of Ownership* π , with respect to $\forall k$ as follows

Definition 1 ($\mathcal{S}_{\text{leeve}}$ [8]) A fallback scheme $\mathcal{S}_{\text{leeve}} = (\text{Gen}_\pi, \text{Sign}, \text{Verify}, \text{Proof}, \text{Verify-Proof})$ is a set of PPT algorithms:

- $\text{Gen}_\pi(1^n)$ on input of a security parameter n outputs a private signing key $s k$, a public verification key $\forall k$ and the back up key $b k$;
- $\text{Sign}(s k, m)$ outputs a signature σ under $s k$ for a message m using the designated main signature scheme, in our example this is an ECDSA signature;
- $\text{Verify}(\forall k, \sigma, m)$ outputs 1 iff σ is a valid signature on m under $\forall k$;
- $\text{Proof}(b k, c)$ on input of the backup information $b k$ and the challenge c , it outputs the ownership proof π . In our example, this is a $W\text{-OTS}^+$ signature on the challenge c using the fallback key $b k$;
- $\text{Verify-Proof}(\forall k, s k, \pi, c)$ is a deterministic algorithm that on input of a public-key $\forall k$, secret-key $s k$, an ownership proof π and a challenge c , it outputs either 0, for an invalid proof, or 1 for a valid one.

The two main security properties of $\mathcal{S}_{\text{leeve}}$ are (1) the capability of issuing a proof to confirm the ownership of the secret key, even in the face of a massive leakage, when

the secret key becomes public, and (2) the capability to smoothly switch to another signature scheme, namely a quantum resistant one. Briefly, we formally review both properties.

Definition 2 (*Proof of Ownership* [8]) For any PPT algorithm \mathcal{A} and security parameter n , it holds

$$\Pr[(\text{vk}, \text{sk}, \text{bk}) \leftarrow \text{Gen}_\pi(1^n) : (c^*, \pi^*) \leftarrow \mathcal{A}(\text{sk}, \text{vk}) \\ \wedge \text{Verify-Proof}(\text{vk}, \text{sk}, \pi^*, c^*) = 1] < \text{negl}$$

for all the probabilities are computed over the random coins of the generation and proof verification algorithms and the adversary.

Definition 3 (*Fallback* [8]) We say that the scheme $(\text{Gen}_\pi, \text{Sign}, \text{Verify})$, with secret and verification key respectively sk and vk such that $\text{Gen}_\pi(1^n) \rightarrow (\text{vk}, \text{sk}, \text{bk})$, has fallback if there are sign and verification algorithms Sign_π and Verify_π such that sk and bk can be used as verification and secret keys respectively, along with Sign_π and Verify_π as fully independent signature scheme.

Tweakable Hash Functions. Introduced to allow better abstraction of hash-based signature scheme. By decoupling the computations of hash chains, hash trees, and nodes, protocol designers can separate the analysis of the high-level construction from exactly how the computation is done. Therefore abstracting the computation away in hash-based schemes only requires analyzing the hashing construction. The standard definition is as follows.

Definition 4 (*Tweakable Hash Function* [3]) Let \mathcal{P} the public parameters space, \mathcal{T} the tweak space, and $n, \alpha \in \mathbb{N}$. A Tweakable Hash Function is an efficient function mapping an α -bit message M to an n -bit hash value MD using a function key called public parameter $P \in \mathcal{P}$ and a tweak $T \in \mathcal{T}$. Therefore, we have $\text{Th} : \mathcal{P} \times \mathcal{T} \times \{0, 1\}^\alpha \rightarrow \{0, 1\}^n$, $\text{MD} \leftarrow \text{Th}(P, T, M)$.

A tweakable hash function takes public parameters P and context information in the form of a tweak T in addition to the message. The public parameters might be thought as a function key or index. The tweak might be interpreted as a nonce. We use the term public parameter for the function key to emphasize it is intended to be public. Thus we explicitly assume an extra property for Th .

Definition 5 (*Indistinguishability*) For the security parameter n , and the tweakable hash function Th , we say that Th has the Computational Indistinguishability from Uniformly Random Distribution Property, if for every PPT distinguisher \mathcal{D} , and arbitrary choices of the parameters P , T and M , the following holds $|\Pr[x \leftarrow \text{Th}(P, T, M), \mathcal{D}(x) = 1] - \Pr[x \leftarrow \mathcal{U}, \mathcal{D}(x) = 1]| \leq \text{negl}(n)$, for the uniform distribution \mathcal{U} .

3 The Tweakable $\mathcal{S}_{\text{Ievee}}$

We now describe our $\mathcal{S}_{\text{Ievee}}$ construction, with W-OTS⁺ as the fallback, and a tweakable hash function for the public key integration, *i.e.* Tweakable $\mathcal{S}_{\text{Ievee}}$.

Definition 6 (*Family of Functions*) Given the security and the Winternitz parameters, respectively, $n \in \mathbb{N}$ and $w \in \mathbb{N}, w > 1$, let a family of functions \mathcal{H}_n be $\{h_k : \{0, 1\}^n \rightarrow \{0, 1\}^n | k \in \mathcal{K}_n\}$ with key space \mathcal{K}_n .

Definition 7 (*Chaining Function*) Given a family of functions $\mathcal{H}_n, x \in \{0, 1\}^n$, an iteration counter $i \in \mathbb{N}$, a key $k \in \mathcal{K}_n$, for j -bit strings $\mathbf{r} = (r_1, \dots, r_j) \in \{0, 1\}^{n \times j}$ with $j \geq i$, then we have the chaining function as follows

$$c_k^i(x, \mathbf{r}) = \begin{cases} h_k(c_k^{i-1}(x, \mathbf{r}) \oplus r_i), & 1 \leq i \leq j; \\ x, & i = 0. \end{cases}$$

Additionally, we review the notation for the subset of randomness vector $\mathbf{r} = (r_1, \dots, r_\ell)$. We denote by $\mathbf{r}_{a,b}$ the subset of (r_a, \dots, r_b) , and for our construction to be presented next, we rely on a Key-Derivation Function KDF which follows the recently announced set of recommendations [9].

Protocol Description. $\mathcal{S}_{\text{Ievee}}$ is 5-tuple set of PPT algorithms (Gen _{π} , Sign, Verify, Proof, Verify-Proof). We describe the generic version of (Gen _{π} , Sign, Verify) in Table 1, based on the formalism of [14] which is convenient for our security analysis in Sect. 4. The algorithms Proof and Verify-Proof are given in Tables 2 and 3, respectively.

Table 1 Gen $\mathcal{S}_{\text{Ievee}}$ is based on the GenElGamal Framework [6, 14] and it relies on the Th which is indistinguishable from the uniform distribution as per Definition 5, and Proof and Verify-Proof are the concrete algorithms

Gen _{π} ^k (1 ⁿ)	Sign ^H (m, sk)	Verify ^H (m, vk, σ)
Pick a random public seed P	$h \leftarrow H(m)$	Parse: $(s, t) \stackrel{p}{\leftarrow} \sigma$
Pick $(\ell + w - 1)$ n -bit strings r_i	$r \stackrel{\$}{\leftarrow} \mathbb{Z}_p; R \leftarrow g^r$	$h \leftarrow H(m)$
Set $\text{bk}_i \leftarrow r_i$, for $1 \leq i \leq \ell$	If $R = 1$: Return \perp	If $(s, h, t) \notin \mathbb{D}$: Return 0
Set $\mathbf{r} = (r_{\ell+1}, \dots, r_{\ell+w-1})$	$t \leftarrow f(R)$	$\hat{R} \leftarrow \mathbf{V}_{g,x}^E(s, h, t)$
Set $\text{vk}_i = c_k^{w-1}(\text{bk}_i, \mathbf{r}), 1 \leq i \leq \ell$	$s \leftarrow \mathbf{S}_{\text{sk}}^E(h, t, r)$	If $\hat{R} = 1$: Return 0
Pick a random hash key \mathcal{X}	If $(s, h, t) \notin \mathbb{D}$: Return \perp	$\hat{t} \leftarrow f(\hat{R})$
Pick a random tweak T	Return $\sigma = (s, t)$	If $t \neq \hat{t}$: Return 0
W-OTS _{pk} ⁺ = Th($P, T, \text{vk}_1, \dots, \text{vk}_\ell$)		Return 1
$\text{sk} \leftarrow ((\mathbf{r}, k), \text{Th}(P, \mathcal{X}, \text{W-OTS}_{\text{pk}}^+))$		
$\text{vk} \leftarrow g^{\text{sk}_1}$		
Return (vk, sk, bk)		

Table 2 Proof algorithm, which is the eW-OTS⁺ Signature Scheme from [8]. The changes introduced by our construction are necessary in order to be used in combination with ECDSA signatures

Proof(c, bk)
Parse $\text{bk} \rightarrow (\text{bk}_0, \text{bk}_1, \dots, \text{bk}_\ell)$
Parse $\text{bk}_0 \rightarrow (\mathcal{T}, \mathcal{P}, \mathcal{X})$
Set $\pi_0 = \text{bk}_0$
Compute $c \rightarrow (c_1, \dots, c_{\ell_1})$,
for $c_i \in \{0, \dots, w-1\}$
Compute checksum $C = \sum_{i=1}^{\ell_1} (w-1-c_i)$,
w -base representation (C_1, \dots, C_{ℓ_2}) ,
for $C_i \in \{0, \dots, w-1\}$
Parse $B = c C$ as $(b_1, \dots, b_{\ell_1+\ell_2})$
Set $\pi_i = c_k^{b_i}(\text{bk}_i, \mathbf{r})$, for $1 \leq i \leq \ell_1 + \ell_2$
Return $\pi = (\pi_0, \pi_1, \dots, \pi_{\ell_1+\ell_2})$

Table 3 The verification of the proof π adapts the verification procedure for eW-OTS⁺ by adding an extra check on the ECDSA verification key vk

Verify-Proof($\text{vk}, \text{sk}, c, \pi$)
Parse $\text{sk} \rightarrow (\text{sk}_0, \text{sk}_1)$
Parse $\text{sk}_0 \rightarrow (\mathbf{r}, k)$
Parse $\pi \rightarrow (\pi_0, \pi_1, \dots, \pi_{\ell_1+\ell_2})$, $\pi_0 \rightarrow (\mathcal{T}, \mathcal{P}, \mathcal{X})$
Compute $c \rightarrow (c_1, \dots, c_{\ell_1})$,
for $c_i \in \{0, \dots, w-1\}$
Compute checksum $C = \sum_{i=1}^{\ell_1} (w-1-c_i)$,
and the base w representation (C_1, \dots, C_{ℓ_2}) ,
for $C_i \in \{0, \dots, w-1\}$
Parse $B = c C$ as $(b_1, \dots, b_{\ell_1+\ell_2})$
Set $\text{vk}_i = c_k^{w-1-b_i}(\pi_i, \mathbf{r}_{b_i+1, w-1})$ for $1 \leq i \leq \ell_1 + \ell_2$
Set $\text{W-OTS}_{\text{pk}}^+ = \text{Th}(P \mathcal{T} \text{vk}_1, \dots, \text{vk}_{\ell_1+\ell_2})$
Return 1, if the following equations hold
$\text{sk}_1 = \text{Th}(P \mathcal{X} \text{W-OTS}_{\text{pk}}^+)$
$\text{vk} = g^{\text{sk}_1}$

3.1 The Generic Sleeve: $\text{GenS}_{\text{leeve}}$

In order to formulate the definition for $\text{GenS}_{\text{leeve}}$, we review more basic definitions to cast it in more generic terms and bases its security on a computational problem, *i.e.* SLP.

Our security analysis relies on the work of [14] which is the state of the art in the understanding of the security of the ECDSA. Their proof bases the analysis in the Semi Logarithm Problem (SLP) with respect to the *Conversion Function* f . Such a

function was introduced in the *GenElGamal Framework* which subsumes ECDSA and other ElGamal based schemes. The proposed framework is parameterized by a *Defining Equation* E for a set \mathbb{D} which gives the distribution of the values to be used in the signature generation, consequently, generating the different “flavors” of the ElGamal/DSA schemes.¹ For a better readability and completeness of this work, we now review these definitions.

Conversion Function. A component of the GenElGamal Framework is the conversion function f . More concretely, the conversation function maps the group members from \mathbb{G} to \mathbb{Z}_q . The SLP is with respect to f and, in its simplest form, can be stated as given a pair of group members g and $X = g^x$, it is required to output s and t such that $t = f((g \cdot X^t)^{\frac{1}{s}})$. Its more general form is given by the next definition.

Definition 8 (SLP [6]) Let (\mathbb{G}, g, q) be a prime-order group and let $f : \mathbb{G}^* \rightarrow \mathbb{Z}_q$ and $\rho_0, \rho_1 : \mathbb{Z}_q^2 \rightarrow \mathbb{Z}_q$ be functions. We say that an algorithm $\mathcal{I}(\tau, \epsilon)$ -breaks the SLP in \mathbb{G} with respect to f , ρ_0 and ρ_1 if it runs in time at most τ and achieves probability $\epsilon = \Pr[X \leftarrow \mathbb{G}; (u, v) \leftarrow \mathcal{I}(g, X) : v = f(g^{\rho_0(u,v)} \cdot X^{\rho_1(u,v)})]$.

The Defining Equation. The sign and verification procedures for the ECDSA and $\mathcal{S}_{\text{leeve}}$ variants can be defined in a modular and general fashion. The technique to make the variants is crucially dependent on the sampling values; Each variant has a different distribution. The Defining Equation rules the distribution, thus we review the definition.

Definition 9 (Defining Equation) Let $\mathbb{D} \subset \mathbb{Z}_q^3$ be a set. An equation $E = E(s, h, t, r, x)$ over $\mathbb{D} \times (\mathbb{Z}_q^*)^2$ is said to be defining (a signature scheme) if E has the form $E(s, h, t, r, x) = C_0(s, h, t) + r \cdot C_r(s, h, t) + x \cdot C_x(s, h, t)$, where C_0 and C_x are functions $\mathbb{D} \rightarrow \mathbb{Z}_q$, and C_r is a function $\mathbb{D} \rightarrow \mathbb{Z}_q^*$. With other words, E is defining if it is affine linear in x and r , and E can always be solved for r .

The concrete example of Defining Equation is $E(s, h, t, r, x) = h - rs + tx$ for the Defining Set $\mathbb{D} = \mathbb{Z}_q^* \times \mathbb{Z}_q \times \mathbb{Z}_q^*$ as given by [14].

Definition 10 (Sign and Verification Function) Let E be a defining equation. Then we define the signing function $\mathbf{S}^E(h, t, r, \text{sk}) = \mathbf{S}_{\text{sk}}^E(h, t, r)$ as follows: if there exists a unique s such that $E(s, h, t, r, \text{sk})$ is satisfied, \mathbf{S}^E returns s ; otherwise, the function returns \perp . Further, we define the verification function $\mathbf{V}^E(g, s, h, t, \text{sk}) = \mathbf{V}_{g, \text{sk}}^E(s, h, t)$ with respect to a prime-order group (\mathbb{G}, g, q) as follows: if r is the (unique) solution of $E(s, h, t, r, \text{sk})$ then \mathbf{V}^E returns g^r .

As remarked by [14], the affine linear form of E makes possible to efficiently evaluate \mathbf{V}^E given just the tuple (s, h, t, g^{sk}) , i.e., without knowing sk explicitly. Now we are ready to define our generic construction.

Definition 11 (Gen $\mathcal{S}_{\text{leeve}}$ Framework) Given a hash function H , and the \mathbf{S} and \mathbf{V} , respectively the Sign and Verification Functions, the Conversion Function f ,

¹ For a complete list of the supported schemes, we refer the reader to the full list in [14].

the Defining Equation E and Set \mathbb{D} , and the Generic $\mathcal{S}_{\text{leeve}}$ scheme is the tuple $(\text{Gen}_{\pi}, \text{Sign}^H, \text{Verify}^H, \text{Proof}, \text{Verify-Proof})$, such that k is the parameter of the family of function, the three first algorithms are given as follows.

4 Security Analysis

This sections introduces the security analysis of the Tweakable $\mathcal{S}_{\text{leeve}}$ in three complementary ways. The next sections cover, respectively, the following:

1. Security with respect of generic attacks and fallback security, as these were introduced in [8];
2. Lemma 1 proposal that shows Tweakable $\mathcal{S}_{\text{leeve}}$ has equivalent security as the ECDSA in terms of unforgeability of signatures, *i.e.* EUF-CMA;
3. The security of the $\text{Gen}\mathcal{S}_{\text{leeve}}$ (introduced in Sect. 3.1), in the same fashion of [14], *i.e.* Generic ECDSA, and show $\text{Gen}\mathcal{S}_{\text{leeve}}$ to be secure with respect to the Semi Logarithm Problem (SLP).

4.1 Generic Attack Security and Unforgeability of Fallback Scheme

The authors of $\mathcal{S}_{\text{leeve}}$ describe in [8] the security level of the construction against generic attacks targeted at the underlying hash function and prove the unforgeability of the fallback scheme. Additionally, they prove that, for an appropriate choice of parameters, the best attack against the fallback scheme (*i.e.*, eW-OTS⁺ the W-OTS⁺ variant introduced in [8]) is the same attack against the original W-OTS⁺. We use these results as a reference as we consider the same fallback scheme and note that, by replacing the assumptions of the underlying hash function with a tweakable hash function, the security results remain well-defined.

4.2 Tweakable $\mathcal{S}_{\text{leeve}}$ is at Least as Secure as an ECDSA One

The security of the ECDSA scheme is given by [14]. However $\mathcal{S}_{\text{leeve}}$ introduces a new key generation method, which is not considered in the security proof of [8]. Concretely, the generation method relies on the tweakable hash function in order to generate the ECDSA secret key sk ; however it is not clear if such modification on the ECDSA scheme introduces weaknesses. We address this gap now.

The Unforgeability of $\mathcal{S}_{\text{leeve}}$. In addition to the listed properties of Sect. 2, $\mathcal{S}_{\text{leeve}}$ is also suitable to similar security definitions as the ones for signature schemes. The

Table 4 Unforgeability for $\mathcal{S}_{\text{leeve}}$, *i.e.* three keys are generated. The above game is One-Message Existential Unforgeability with Chosen Message Attack game, *i.e.* (EUF-CMA1). For the general form, *i.e.* the standard (EUF-CMA), the Sign Procedure does not abort when the message is in the list \mathcal{L} . For the key only (UF-KOA) variant of the game, the adversary does not access the Sign Procedure

Procedure Init(n) $\mathcal{L} \leftarrow \emptyset$ $(\mathsf{vk}, \mathsf{sk}, \mathsf{bk}) \leftarrow \text{Gen}_{\pi}(1^n)$ Return vk Procedure Fin(m^*, σ^*) If $m^* \in \mathcal{L}$: Abort If $\text{Verify}(\mathsf{vk}, m^*, \sigma^*) = 0$: Abort Return 1	Procedure Sign(m) If $m \in \mathcal{L}$: Abort $\sigma \leftarrow \text{Sign}(\mathsf{sk}, m)$ If $\sigma = \perp$: Return \perp $\mathcal{L} \leftarrow \mathcal{L} \cup \{m\}$ Return σ
--	--

difference is the generation of the keys, which $\mathcal{S}_{\text{leeve}}$ introduces an extra one, the back up key. Table 4 defines the security notion, derived from standard EUF-CMA security for signature schemes. The difference is only the extra back up key.

The goal of the next lemma is to show that the EUF-CMA security of $\mathcal{S}_{\text{leeve}}$, constructed with a suitable tweakable hash function, and ECDSA, instantiated with uniformly random sampling for the secret key, are equivalent.

Lemma 1 *Assume ECDSA is EUF-CMA secure and the generation algorithm Gen_{π} from Table 1 is constructed with a tweakable hash function Th indistinguishable from the uniform distribution as per Definition 5 for the security parameter n . Then $\mathcal{S}_{\text{leeve}}$ is EUF-CMA as given by the security game of Table 4.*

Proof (sketch) Assume the existence of a $\mathcal{S}_{\text{leeve}}$ forger \mathcal{F} which wins the game from Table 4 by outputting a forgery (m^*, σ^*) with non-negligible probability. Then we construct a PPT distinguisher algorithm \mathcal{D} which breaks the indistinguishability property of Th with high probability. We construct \mathcal{D} as follows:

- \mathcal{D} performs the security game given by Definition 5, and receives as input the string x ;
- \mathcal{D} modifies the generation algorithm Gen_{π} from Table 1, by using the received string x to generate the public key. In the modified game the public key is $\mathsf{vk}' = g^{\mathsf{sk}'}$ for $\mathsf{sk}' \leftarrow x$;
- \mathcal{D} simulates the EUF-CMA security game of Table 4 to \mathcal{F} using $(\mathsf{vk}', \mathsf{sk}')$;
- With high probability \mathcal{F} outputs (m^*, σ^*) , then \mathcal{D} uses the verification algorithm Verify to perform the following and stop:
 - If $\text{Verify}(m^*, \sigma^*) = 1$, then output 1
 - Else, output 0;

Now we estimate the success probability of \mathcal{F} in the EUF-CMA game of Table 4, by considering three points:

- From the indistinguishability property of Th , we know $|Pr[x \leftarrow \text{Th}(P, T, M), \mathcal{D}(x) = 1] - Pr[x \leftarrow \mathcal{U}, \mathcal{D}(x) = 1]|$ is negligible for arbitrary choices of P , T and M as given by Definition 5 and initial hypothesis;
- Following from the EUF-CMA security of ECDSA, we have that $Pr[x \leftarrow \mathcal{U}, \text{Verify}(m^*, \sigma^*) = 1]$ is negligible for the uniform random distribution \mathcal{U} ;
- Finally, note that $Pr[x \leftarrow \text{Th}(P, T, v), \mathcal{D}(x) = 1]$ and $Pr[x \leftarrow \text{Th}(P, T, v), \text{Verify}(m^*, \sigma^*) = 1]$ are equal by design of \mathcal{D} and success probability of \mathcal{F} .

Therefore, $|Pr[x \leftarrow \text{Th}(P, T, M), \mathcal{D}(x) = 1] - Pr[x \leftarrow \mathcal{U}, \mathcal{D}(x) = 1]| \leq \text{negl}(n)$, and $|Pr[x \leftarrow \text{Th}(P, T, M), \text{Verify}(m^*, \sigma^*) = 1] - \text{negl}(n)| \leq \text{negl}(n)$. Hence $Pr[x \leftarrow \text{Th}(P, T, v), \text{Verify}(m^*, \sigma^*) = 1]$ must be negligible and $\mathcal{S}_{\text{leeeve}}$ is also EUF-CMA, thereby giving the lemma. \square

The earlier lemma only relates the security of $\mathcal{S}_{\text{leeeve}}$ and ECDSA. In order to thoroughly prove the hardness of breaking $\mathcal{S}_{\text{leeeve}}$ it is convenient to consider a computational problem. That is what we do next.

4.3 The Security of $\text{Gen}\mathcal{S}_{\text{leeeve}}$

From now we take the approach of [14] in order to build a full proof of the unforgeability of the generic $\mathcal{S}_{\text{leeeve}}$ variant based on the assumed hard computational problem. Namely, show the security of $\text{Gen}\mathcal{S}_{\text{leeeve}}$ with respect to SLP. What we do now is to review the main definitions from [14] combined with the ones introduced in Sect. 3.1.

Definition 12 (*h-decomposable*) Let $E = E(s, h, t, r, x)$ be a defining equation with corresponding set \mathbb{D} . We say that E is *h-decomposable* (with respect to \mathbb{D}) if there exist functions $v_0, v_1 : \mathbb{Z}_q \rightarrow \mathbb{Z}_q$ and $\rho_0, \rho_1 : \mathbb{Z}_q^2 \rightarrow \mathbb{Z}_q$ such that $v_0, v_1 \neq 0$ if $h \neq 0$ and $r = v_0(h) \cdot \rho_0(s, t) + x \cdot v_1(h) \cdot \rho_1(s, t)$ for all $(s, h, t) \in \mathbb{D}$ and $r, x \in \mathbb{Z}_q^*$ satisfying $E(s, h, t, r, x)$.

For completeness, in the next definition we consider the standard notion for δ statistical distance. That is, for any two ensembles $\{X(x, k)\}_{x \in \{0,1\}^*, k \in \mathbb{N}}$ and $\{Y(x, k)\}_{x \in \{0,1\}^*, k \in \mathbb{N}}$, for index k and input x , the value $|\Pr[X(x, k) = 1] - \Pr[Y(x, k) = 1]|$ is at most δ .

Definition 13 (*δ -Simulatability*) Let $(E, \mathbb{G}, H, f, \mathbb{D})$ be an instantiation of $\text{Gen}\mathcal{S}_{\text{leeeve}}$ as in Definition 11. It is said that the instantiation is δ -simulatable if there exists a function $\text{Sim}^E : \mathbb{Z}_q^3 \times \mathbb{Z}_q^2 \cup \{\perp\}$ that is computable in about the same time as \mathbf{S}^E such that for all $s, k \in \mathbb{Z}_q^*$ the statistical distance between the outputs of the two protocols depicted by Table 5 is at most δ .

The generic security is derived from the work on [14]. Namely, the next two theorems which are defined according to the number of random oracle and signature queries, respectively \mathcal{Q}_H and \mathcal{Q}_s and the big-O notation \mathcal{O} . For completeness we present them altered to $\text{Gen}\mathcal{S}_{\text{leeeve}}$. However we refer the reader to the full work for the proofs of the theorems, which are the same for $\text{Gen}\mathcal{S}_{\text{leeeve}}$.

Table 5 \mathcal{P}_{Sim} shows that, given a procedure Sim , it is possible to generate a tuple (s, h, t) statistically close without knowing the secret key sk

$\mathcal{P}_{\text{real}}(\text{sk}, g)$	$\mathcal{P}_{\text{Sim}}(\text{vk}, g)$
$r \xleftarrow{\$} \mathbb{Z}_p$	$a, b \xleftarrow{\$} \mathbb{Z}_p$
$R \leftarrow g^r$	$R \leftarrow \text{vk}^a g^b$
If $R = 1$: Return \perp	If $R = 1$: Return \perp
$t \leftarrow f(R)$	$t \leftarrow f(R)$
$h \xleftarrow{\$} \mathbb{Z}_q$	$(s, h) \leftarrow \text{Sim}^E(a, b, t)$
$s \leftarrow \mathcal{S}_{\text{sk}}^E(h, t, r)$	If $(s, h, t) \notin \mathbb{D}$: Return \perp
If $(s, h, t) \notin \mathbb{D}$: Return \perp	Return (s, h, t)
Return (s, h, t)	

Theorem 1 [14] *Let $(E, \mathbb{G}, H, f, \mathbb{D})$ be a δ -simulatable of $\text{Gen}\mathcal{S}_{\text{leeve}}$. Then if H is modeled as random oracle, for every forger \mathcal{F} that $(\tau, \mathcal{Q}_s, \mathcal{Q}_H, \varepsilon)$ -breaks the one-per-message unforgeability if this instantiation there also exists a forger \mathcal{F}' that $(\tau', 0, \mathcal{Q}_H, \varepsilon')$ -breaks the key-only unforgeability of this instantiation, where $\varepsilon' \geq \varepsilon / (e^2(\mathcal{Q}_s + 1)) - \mathcal{Q}_s \delta$ and $\tau' = \tau + \mathcal{O}(\mathcal{Q}_H)$.*

Theorem 2 [14] *Let (\mathbb{G}, g, q) be a prime-order group, let E be a defining equation with corresponding set \mathbb{D} , and let $f : \mathbb{G}^* \rightarrow \mathbb{Z}_q$ and $H : \{0, 1\}^* \rightarrow \mathbb{Z}_q$ be functions. If E is h -decomposable with functions ρ_0 and ρ_1 , and H is modelled as a random oracle, then the SLP in \mathbb{G} with respect to f, ρ_0, ρ_1 is non-tightly equivalent to the key-only unforgeability of $\text{Gen}\mathcal{S}_{\text{leeve}}$ when instantiated with $(E, \mathbb{G}, H, f, \mathbb{D})$.*

That is, for any adversary \mathcal{I} that (τ, ε) -breaks SLP, there exists a forger \mathcal{F} that (τ', ε) -breaks the key-only unforgeability of Generic Sleeve, where $\tau \approx \tau'$.

Conversely, for any forger \mathcal{F} that $(\tau, \mathcal{Q}_H, \varepsilon)$ -breaks the key-only unforgeability of $\text{Gen}\mathcal{S}_{\text{leeve}}$, there exists an adversary \mathcal{I} that $(\tau', \varepsilon / \mathcal{Q}_H - 1/q)$ -breaks SLP, where $\tau \approx \tau'$ and \mathcal{Q}_H is the number of random oracle queries posed by \mathcal{F} .

Sections 4.1, 4.2 and 4.3 fully cover the security of the Tweakable $\mathcal{S}_{\text{leeve}}$, regarding ECDSA security, and $\text{Gen}\mathcal{S}_{\text{leeve}}$ with respect to SLP.

We now focus on the experimental results.

5 Implementation and Performance

This section describes our open-source implementation, the audit results along with the associated fixes, and details of the Verifpal formal analysis model.

Reference Implementation. We implemented a single-threaded version in Golang. In our implementation, W-OTS^+ uses SHA3 for public key compression and Blake2b for hash ladder calculations. We use the secp256k1 curve with ECDSA as

the main signature scheme and verified the correctness of our code, which integrates BIP39 [4], by comparing it with reference BIP39 implementations [5, 15]. Our implementation differs slightly from the original W-OTS⁺ specification, which defines a secret key as ℓ random numbers and, instead, derives the secret key values from a single *seed* parameter by using a KDF. The W-OTS_{pk}⁺ is compressed using a tweakable hash function using the public seed, and the secret hash key value \mathcal{X} .

Audit. We expose the detailed results obtained from the official audit of the reference implementation and the subsequent fixes.

- **Scope:** The scope of the audit included the correctness of the cryptography and associated security, finding eventual timing leaks, usage of unsafe APIs, missing security checks, risk from dependencies, and poor randomness generation.
- **Security Issues:** No outstanding security issue appeared in the core cryptographic modules and the main security remarks are associated with a command line interface (CLI) tool created to improve the usability of the user. The audit results are openly available in [23].
- **Verifpal Implementation:** The code associated with the formal analysis tools is openly available on a Github repository in a special folder dedicated to the formal verification component [23].

Performance Metrics. We present performance metrics for our single-threaded implementation running on one Amazon c5.xlarge benchmark machine with an Intel Xeon Platinum 8124M 3.00GHz CPU and 8GiB RAM. Our code runs a $\mathcal{S}_{\text{leeve}}$ key generation in 1.81 ms, which comprises a W-OTS⁺ key generation that takes 1.75 ms and an ECDSA key generation that takes 0.059 ms. These early results demonstrate that the key generation of the (tweakable) $\mathcal{S}_{\text{leeve}}$ construction is significantly slower than presently used key generation mechanism (i.e., ECDSA). These results are expected as the $\mathcal{S}_{\text{leeve}}$ construction introduces a significant amount of additional steps in the wallet generation process. Potential improvements may include calculating the W-OTS⁺ hash-ladders in parallel and the use of different and potentially faster hash functions implementations.

6 Formal Methods Analysis

This section reports on the mathematical security proof of our construction, and outlines the Verifpal [20, 21] model we used to analyze the tweakable $\mathcal{S}_{\text{leeve}}$ along with some of the challenges that appeared throughout this process. We start by giving a brief summary on the Verifpal tool.

Verifpal. Verifpal is a software for verifying the security of cryptographic protocols. This tool is oriented towards real-world practitioners attempting to integrate formal verification into their line of work. To achieve this, Verifpal uses a new, intuitive language for modeling protocols that is considered easier to write and understand than the languages currently employed by existing tools.

Challenges to Modelling $\mathcal{S}_{\text{leeve}}$ in Verifpal. A commonly found problem in symbolic model protocol verifiers is that, for complex protocols, the different combinations of variables that the verifier must assess, quickly becomes too large to terminate in reasonable time. This is a challenge we faced in our modelling process as we initially attempted to model a W-OTS^+ fallback for ECDSA and the tool constantly issued memory fault errors when starting to perform the hash ladder iterations, which resulted in the stopping of the verification process in a faulty manner. Additionally, we highlight the lack of existence of the XOR logical function in the tool, which lead to design attempts with changed variants of the chaining function.

Verifpal Model of $\mathcal{S}_{\text{leeve}}$. To avoid the memory fault issues derived from iterating different attack scenarios involving a high number of hash function calls, we model a simpler Lamport signature scheme as a quantum-secure fallback instead of W-OTS^+ .

Attacker model. All the interactions in the model go through an active attacker. Therefore, we assume the Dolev-Yao model [12] where the adversary is in charge of delivering the messages.

Results. The tool output that regardless of the compromise of the ECDSA secret key value, the queried values remain confidential, and only the true owner of the hash-based fallback key pair is able to produce a safety signature. We assume correctness of the Verifpal execution results, especially since there results match the results obtained in the security proof of $\mathcal{S}_{\text{leeve}}$.

7 Final Remarks

The $\mathcal{S}_{\text{leeve}}$ definition is a promising and novel scheme designed as an extension to existing wallets since, as quantum computers evolve, the security of most cryptocurrency wallets is at risk.

In this work, we improve on the original $\mathcal{S}_{\text{leeve}}$ construction by proposing the Tweakable $\mathcal{S}_{\text{leeve}}$. Thus we introduced a more modular approach that is simpler to analyze and implement. Moreover, we fill the missing gaps in the security proof of the original proposal, connecting it to the state-of-the-art of the ECDSA security. Namely, (1) our construction presents the same capabilities of the original $\mathcal{S}_{\text{leeve}}$, (2) it is at least as secure as the ECDSA signature scheme given a tweakable hash function whose output is computationally indistinguishable from the uniformly random distribution, and (3) our construction is generically secure, *i.e.* $\text{Gen}\mathcal{S}_{\text{leeve}}$ with respect to SLP.

Finally, we showcase our security results using the formal method analysis tool called *Verifpal*, which produced positive results matching the ones obtained in the mathematical proof of security. The distinctive extra level of security provided by $\mathcal{S}_{\text{leeve}}$ has the potential to help in the adoption of this new cryptographic primitive in the context of blockchain applications. Moreover this work illustrates that our construction is now open-source, audited, and features a more complete security

proof relating the construction with a concrete computational problem: a *must* in provable security practice.

Finally, this work illustrates a fruitful combination of theoretical work, from the protocol specification/construction, to the formal method analysis. Such thorough work which might raise the expectation of due diligence teams to include formal analysis when designing and evaluating cryptographic protocols.

A High-level Diagram of the Tweakable $\mathcal{S}_{\text{leeve}}$ Construction

This section exposes a high-level diagram of the sequence of performed steps in the key generation component of the $\mathcal{S}_{\text{leeve}}$ construction (Fig. 2).

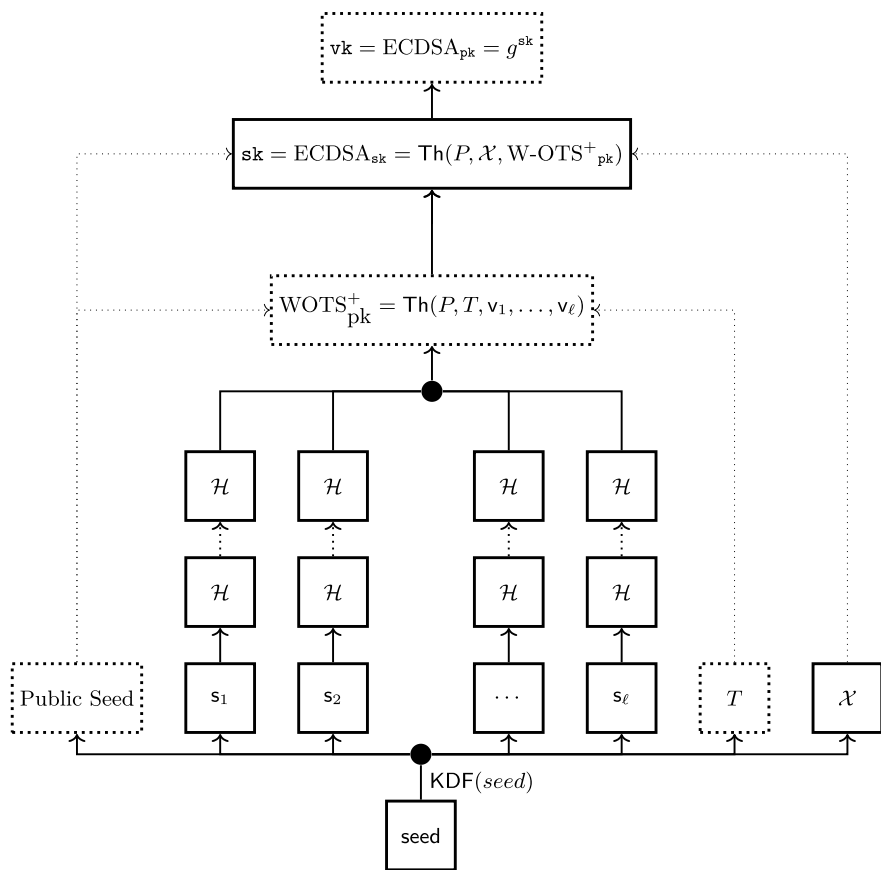


Fig. 2 $\mathcal{S}_{\text{leeve}}$ high-level diagram of the key generation

References

1. Aranha, D. F., Novaes, F. R., Takahashi, A., Tibouchi, M., & Yarom, Y. (2020). Ladderleak: Breaking ecDSA with less than one bit of nonce leakage. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security* (pp. 225–242). New York, NY, USA: CCS '20, Association for Computing Machinery.
2. Badertscher, C., Gazi, P., Kiayias, A., Russell, A., & Zikas, V. (2018). Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In D. Lie, M. Mannan, M. Backes, & X. Wang (Eds.), *ACM CCS* (pp. 913–930). ACM Press. <https://doi.org/10.1145/3243734.3243848>.
3. Bernstein, D. J., Hülsing, A., Kölbl, S., Niederhagen, R., Rijneveld, J., & Schwabe, P. (2019). The SPHINCS⁺ signature framework. In L. Cavallaro, J. Kinder, X. Wang, & J. Katz (Eds.), *ACM CCS* (pp. 2129–2146). ACM Press. <https://doi.org/10.1145/3319535.3363229>.
4. Mnemonic code for generating deterministic keys. Accessed September 10, 2021, from <https://github.com/bitcoin/bips/blob/master/bip-0039/mediawiki>.
5. Mnemonic code converter. Accessed September 10, 2021, from <https://iancoleman.io/bip39/>.
6. Brown, D. (2005). On the provable security of ECDSA, pp. 21–40. London Mathematical Society Lecture Note Series, Cambridge University Press.
7. Brown, D. R. (2005). Generic groups, collision resistance, and ecDSA. vol. 35, pp. 119–152. Springer.
8. Chaum, D., Larangeira, M., Yaksetig, M., & Carter, W. (2021). Wots+ up my sleeve! a hidden secure fallback for cryptocurrency wallets. In *International Conference on Applied Cryptography and Network Security* (pp. 195–219). Springer.
9. Chen, L. (2022). Recommendation for key derivation using pseudorandom functions-revision 1. NIST special publication. Accessed February 20, 2022, from <https://doi.org/10.6028/NIST.SP.800-108r1-draft>.
10. Dahmen, E., Okeya, K., Takagi, T., & Vuillaume, C. (2008). Digital signatures out of second-preimage resistant hash functions. In J. Buchmann, & J. Ding (Eds.), *Post-quantum Cryptography, Second International Workshop, PQCRYPTO* (pp. 109–123). Heidelberg: Springer. https://doi.org/10.1007/978-3-540-88403-3_8.
11. David, B., Gazi, P., Kiayias, A., & Russell, A. (2018). Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In: J. B. Nielsen, & V. Rijmen (Eds.), *EUROCRYPT, Part II. LNCS* (vol. 10821, pp. 66–98). Heidelberg: Springer. https://doi.org/10.1007/978-3-319-78375-8_3.
12. Dolev, D., & Yao, A. (1983). On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2), 198–208.
13. Fersch, M., Kiltz, E., & Poettering, B. (2016). On the provable security of (ec)dsa signatures. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security* (pp. 1651–1662). New York, NY, USA: CCS '16, Association for Computing Machinery.
14. Fersch, M., Kiltz, E., & Poettering, B. (2017). On the one-per-message unforgeability of (EC)DSA and its variants. In: Y. Kalai, & L. Reyzin (Eds.), *TCC 2017, Part II. LNCS* (vol. 10678, pp. 519–534). Heidelberg: Springer https://doi.org/10.1007/978-3-319-70503-3_17.
15. Golang implementation of the bip39 spec. Accessed September 10, 2021, from <https://godoc.org/github.com/tyler-smith/go-bip39>.
16. Hülsing, A. (2013). W-OTS+ - shorter signatures for hash-based signature schemes. In A. Youssef, A. Nitaj, & A. E. Hassanien (Eds.), *AFRICACRYPT 13. LNCS* (vol. 7918, pp. 173–188). Heidelberg: Springer. https://doi.org/10.1007/978-3-642-38553-7_10.
17. Ilie, D. I., Karantias, K., & Knottenbelt, W. J. (2020). Bitcoin crypto-bounties for quantum capable adversaries. *Cryptology ePrint Archive*, Paper 2020/186. <https://eprint.iacr.org/2020/186>.
18. Ilie, D. I., Knottenbelt, W. J., & Stewart, I. (2020). Committing to quantum resistance, better: A speed-and-risk-configurable defence for bitcoin against a fast quantum computing attack. *Cryptology ePrint Archive*, Paper 2020/187. <https://eprint.iacr.org/2020/187>.

19. Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In: J. Katz, & H. Shacham (Eds.), *CRYPTO 2017, Part I. LNCS* (vol. 10401, pp. 357–388). Heidelberg: Springer. https://doi.org/10.1007/978-3-319-63688-7_12.
20. Kobeissi, N. (2021). Verifpal: Cryptographic Protocol Analysis for Students and Engineers. Accessed August 5, 2021, from <https://verifpal.com>.
21. Kobeissi, N., Nicolas, G., & Tiwari, M. (2020). Verifpal: Cryptographic protocol analysis for the real world. In *Proceedings of the 2020 ACM SIGSAC Conference on Cloud Computing Security Workshop* (p. 159). New York, NY, USA: CCSW'20, Association for Computing Machinery.
22. Nakamoto, S. (2009). Bitcoin: A peer-to-peer electronic cash system. <http://www.bitcoin.org/bitcoin.pdf>.
23. Sleeve. (2022). Accessed February 21, 2022, from https://github.com/xx-labs/sleeve/tree/main/verifpal_model.
24. Trinity attack incident part 1: Summary and next steps. Accessed September 22, 2020, from <https://blog.iota.org/trinity-attack-incident-part-1-summary-and-next-steps-8c7ccc4d81e8>.
25. Wood, G. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151, 1–32.

Interhead Hydra: Two Heads are Better than One



Maxim Jourenko, Mario Larangeira, and Keisuke Tanaka

Abstract Distributed ledger are maintained through consensus protocols which have inherent limitations to their scalability. Layer-2 protocols operate on *channels* and allow parties to interact with another without going through the consensus protocol albeit relying on its security as fall-back. Channels can be concatenated into networks using techniques such as Hash Timelock Contracts (HTLC) to execute payments or virtual state channels as introduced by Dziembowski et al. [CCS'18] to execute state machines across a path of channels. This is realized by utilizing *intermediaries*, which are the parties on the channel path between both endpoints, who have to pay collateral to ensure security of the constructions. Dziembowski et al. [Eurocrypt'19] introduced multi-party state channels based on a virtual channel construction and more recently Hydra *heads* [FC'21] is a channel construction that allows multiple parties the execution of Constraint Emitting Machines (CEM). While existing protocols such as HTLCs can be extended such that two parties can interact with another across channels, there are no dedicated constructions that utilize multi-party channels and similarly allow more than two parties to interact across a network of channels. This work addresses this gap by extending Hydra and introducing the Interhead construction that allows for the iterative creation of virtual Hydra heads.

This work was supported by JST CREST JPMJCR2113 and JSPS KAKENHI 21H04879.

M. Jourenko (✉)

Department of Mathematical Informatics, Graduate School of Information Science and Technology, The University of Tokyo, Tokyo, Japan
e-mail: jourenko.m.ab@m.titech.ac.jp

M. Larangeira · K. Tanaka

Department of Mathematical and Computing Sciences, School of Computing, Tokyo Institute of Technology, Tokyo, Japan
e-mail: mario@c.titech.ac.jp; mario.larangeira@iohk.io
URL: <http://iohk.io>

K. Tanaka

e-mail: keisuke@is.titech.ac.jp

M. Larangeira

Input Output Hong Kong, Hong Kong, China

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_11

Our construction is the first that (1) supports and utilizes multi-party channels and (2) allows for collateral to be paid by multiple intermediaries.

Keywords Blockchain · State channel · Channel network

1 Introduction

Decentralized ledger were first introduced by Nakamoto [20] with the blockchain technology. Cryptocurrencies that are based on this design enjoy a steadily increasing popularity since then. However, while a wider adaption of decentralized ledger shows the relevance of this technology, the existing implementations struggle to be scalable to the increased demand. Transactions on a decentralized ledger require to be processed through a consensus mechanism, which are classified as Layer-1 protocols. The transaction throughput that these protocols permit is the limiting factor on a ledger's scalability, and increasing it is non-trivial [5]. Issuers of a transaction can pay fees to increase the priority under which it is processed. This results in the creation of a marketplace where processing a transaction requires payment of an amount of fees that correlates with the demand for processing transactions. For instance, on 20th April 2021 the average cost of processing a transaction in Ethereum peaked at more than 71\$.¹

Layer-2 protocols are a classification of techniques that aim to reduce the number of transactions that are issued on a ledger by means of a layer of indirection. Approaches include sidechains [10, 24], Ethereum's Plasma [23], payment channels [6, 21, 22] and more generally state channels [8, 9] notably multi-party state channels such as Multi-Party Virtual State Channels (MPVSC) [7] and Hydra *heads* [4]. Payment channels are setup by two parties using a transaction that locks their coins into a shared wallet. The channel stores a state which is how the funds that are locked inside it are distributed between both parties. The parties can then perform an *offchain* protocol to change this state. Lastly, the parties issue one transaction to unlock the coins from the channel corresponding to its latest state. Note that the latter transaction is created first to avoid that coins are locked in the channel indefinitely if one of the parties is unresponsive. This way parties can perform $\mathcal{O}(n)$, $n \in \mathbb{N}$, payments with each other while only issuing $\mathcal{O}(1)$ transactions on the ledger thus improving the system's scalability. State channels extend this notion by allowing for the storage of arbitrary state which, with the conjunction of a sufficiently expressive scripting mechanism, can allow the two parties to execute state machines offchain. MPVSC and Hydra generalize the notion of channels further by allowing an arbitrary amount of parties to participate in one channel. Offchain protocols like Hash Timelock Contracts (HTLCs) [22] and virtual channels [9, 12, 13] allow adjacent channels, i.e. channels with one common party, to be concatenated into channel net-

¹ https://ycharts.com/indicators/ethereum_average_transaction_fee.

works which allow parties to interact with another by reusing the existing channel infrastructure.

Whereas protocols for performing payments or execution of state machines across a channel network exist there is no analogous protocol for multi-party channels such as Hydra. Although HTLCs can be reused to perform offchain payments across Hydra heads, there is no way to execute arbitrary smart contracts offchain across multiple Hydra heads. Additionally, existing approaches to create virtual channels across multiple underlying channels [9, 12] rely on one *intermediary* who can ensure correctness and therefore can be uniquely blamed in case of a fault, and moreover, the intermediary commits a *collateral* which can be used to refund any potential loss of a honest party. However, it is not trivial to extend these approaches to the case of multi-party channels.

Layer-2 structures move interactions off-chain thus minimizing expensive on-chain transactions. In the case of Hydra this is done without compromising on the expressiveness of transactions due to its isomorphic property: Interactions that are possible on-chain are permitted within Hydra heads. The aim of this work is to allow parties across multiple Hydra heads to have that same degree of interaction, while remaining offchain, i.e. no interaction with the ledger is required except in case of dispute. Similar to related offchain protocols [8, 9, 12] this improves scalability by enabling reuse of existing offchain infrastructure.

Related Work. Payment channels are one of the earliest Layer-2 protocols that aim to improve scalability of a decentralized ledger by moving transactions offchain while relying on the ledger to resolve disputes. To create a channel two parties collaborate to lock coins on the ledger such that these coins can only be spent with the consent of both parties. In advance both parties prepare a **Refund** transaction that spends the coins locked within the channel and returns both parties' coins respectively which closes the channel. Instead of committing this transaction to the ledger, both parties can perform payments among another by computing another Refund transaction, where the channel's coins are distributed differently, while invalidating all older Refund transactions. Channel constructions differ in how Refund transactions are invalidated [6, 21, 22]. While payment channels can be implemented on ledgers with simple scripting languages such as Bitcoin [20], constructions that make use of smart contracts allow for the creation of state channels [8, 9] that in addition allow execution of smart contracts within Layer-2. Channels can be extended into channel networks for which there are two families of approaches. For one, channels can be concatenated into channel networks, by allowing for payments across multiple hops of an underlying channel network [13, 19, 22] where a prominent example is through the use of HTLCs [22]. Another approach is to create new *virtual* channels offchain [8, 9, 12]. Note that these virtual channel constructions have a limited lifetime which is fixed upon setup. While the previous channel constructions are always between two parties, further work was done to allow creation of channels between an arbitrary amount of parties as MPVSC [7] and Hydra *heads* [4]. Notably, Hydra heads maintain a subset of the ledger's state such that parties within a Hydra head can interact with another the same as on the ledger itself. As of yet there are no protocols

that allow to extend these multi-party channels into channel networks while making use of the properties of multi-party channels.

Our Contributions. This work extends Hydra [4], which is the computational layer of the Cardano ledger. We propose a protocol to create a *virtual* Hydra head on top of two existing Hydra heads. Our approach consists of two components, an Interhead State Machine, which is derived from the Hydra state machine, as well as a protocol that defines the parties' behaviour. The construction (1) can be executed iteratively to form a virtual Hydra head across a network of Hydra heads, (2) has no limit on it's lifetime, (3) allows for the presence of multiple intermediaries who can share the burden of committing the required collateral, (4) is a Layer-2 protocol, i.e. if all parties collaborate no transactions are added to the ledger, (5) is secure in the presence of a malicious static adversary, (6) does not put honest parties at risk of loss of their coins. While this work focuses on the creation of a virtual channel for Hydra heads, we argue that the state machine proposed in this work can be implemented for other channel constructions such as [7] on blockchains with sufficiently expressive smart contracts such as Ethereum. As of such we present the, to our knowledge, first virtual channel construction with properties (1)–(3) from above which we think is of independent interest.

2 Background

Notation. In this work we make frequent use of tuples to structure data. Let α be an instance of a tuple of type \mathcal{A} of form $(\alpha_0, \dots, \alpha_n)$, $n \in \mathbb{N}$ where $\alpha_0, \dots, \alpha_n$ are the entries' labels. Then we address entry $i \in \mathbb{N}$, $0 \leq i \leq n$ of α using its name and the entry's label, i.e. $\alpha.\alpha_i$. We denote \mathbb{N} as the set of natural numbers.

Signature Schemes. We assume the existence of two secure digital signature schemes which both fulfill notions of **completeness** and **unforgeability**, however, we remain rather informal in the remainder. First, we assume a signature scheme [1] consisting of algorithms (`key_gen`, `verify`, `sign`) such that `key_gen`(1^λ) = (vk, sk) creates a pair of secret key sk and verification key vk under security parameter λ , `sign`(sk, m) = σ creates a signature such that `verify`(vk, m, σ') evaluates to `True` if and only if $\sigma' = \sigma$. Second, we assume the existence of a multi-signature scheme [11, 18] of form (`ms_setup`, `ms_key_gen`, `ms_agg_vk`, `ms_sign`, `ms_agg_sign`, `ms_verify`) where `ms_setup`($1^{\lambda'}$) = Π creates public parameter Π with security parameter λ' , `ms_key_gen`(Π) = (vk', sk') creates a key pair consisting of secret key sk and verification key vk' , `ms_agg_vk`(Π, V) = avk aggregates a set of verification keys V into an aggregate verification key avk , `ms_sign`(Π, sk', m') = σ'' creates a signature σ'' of message m corresponding to secret key sk' whereas `ms_agg_sign`(Π, V, S, m') = σ_{agg} aggregate a set of signatures S into aggregate signature σ_{agg} such that `ms_verify`(Π, m', avk, σ'') evaluates to `True` if and only if $\sigma'' = \sigma_{agg}$ and evaluates to `False` otherwise.

The EUTxO Model. The Extended Unspent Transaction Outputs (EUTxO) model was introduced by Chakravarty et al. [2] and it improves on the UTxO paradigm as used with ledgers such as Bitcoin [20] by allowing for the execution of smart contracts defined as Constraint Emitting Machines (CEMs), thus improving the system's expressiveness. An EUTxO based Ledger's state consists of a set of EUTxO $S \subseteq \text{EUTxO}$ where EUTxO is the set of all possible EUTxO. More specifically, it consists of a set of (out_{ref}, u) where u is a EUTxO representing coins that are in circulation, and out_{ref} is a unique identifier that can be used to reference u and is commonly derived from the context it was created in. A transaction tx is a tuple of form (I, O, r, S) where I is a set of inputs, i.e. entries of form (out_{ref}, u) , O is a list of outputs, i.e. newly defined EUTxO, r is a validity interval, i.e. $[r_0, r_1]$ where $r_0, r_1 \in \mathbb{N}$ are points in time, and S is a set of signatures. If a transaction is sent to the ledger within the interval r , the amount of coins in O is at least as large as the amount of coins referenced in I and all validity scripts evaluate to True , the transaction induces a state transition on the ledger by removing all entries in I from its state and adding the newly defined EUTxO in O to the ledger's state. A transaction is processed by the ledger within time $\Delta \in \mathbb{N}$. Note that a transaction that is applied on the ledger has to be processed by a consensus protocol which requires payment of a fee. A EUTxO u itself is a tuple of form $(v, \text{value}, \delta)$ where $v \in \{0, 1\}^*$ is a validator script written in a Turing complete language, $\text{value} \in \mathbb{N}$ is an amount of coins, and $\delta \in \{0, 1\}^*$ is arbitrary data. An EUTxO can be spent making its coins accessible, if a party can show a redeemer value $\rho \in \{0, 1\}^*$ such that $v(\rho, \delta, \sigma) = \text{True}$, where σ is the validation context that includes information on the transaction that spends u as well as all EUTxO referenced in its inputs.

Constraint Emitting Machines. Chakravarty et al. [2] showed a weak bi-simulation between programs running on the EUTxO ledger and Constraint Emitting Machines (CEMs) derived from Mealy machines [17]. Thus, CEMs can be used to define applications for an EUTxO ledger. A CEM is a tuple $(\mathbf{S}, \mathbf{I}, \text{step}, \text{initial}, \text{final})$ where \mathbf{S} is a possibly infinite set of states, \mathbf{I} is a set of input symbols, **initial**, **final** are functions $\mathbf{S} \rightarrow \mathbb{B}$ indicating initial and final states respectively and function **step** : $\mathbf{S} \rightarrow \mathbf{I} \rightarrow \text{Maybe}(\mathbf{S}, \text{TxConstraints})$ is a partial function that maps to a new state with constraints **TxConstraints**.

Hydra Heads. Hydra is a scalability solution for EUTxO based ledger. It extends the idea of channels by (1) being constructed between an arbitrary amount of parties and (2) being isomorphic, i.e. allowing all transactions that are permitted on the ledger. Hydra allows an arbitrary set of parties to take a set of EUTxO $\eta_0 \subseteq S$ offchain where η_0 is called a snapshot and S is the ledger's state as described above. When open, the parties perform an offchain consensus protocol among another to modify η_0 by applying transactions to it which results in a new snapshot η_1 . This can be repeated to create snapshots $\eta_2, \dots, \eta_n, n \in \mathbb{N}$. The Hydra head is closed by moving η_n back into the ledger's state. The simplified Hydra CEM as shown in Fig. 1 has four states initial,

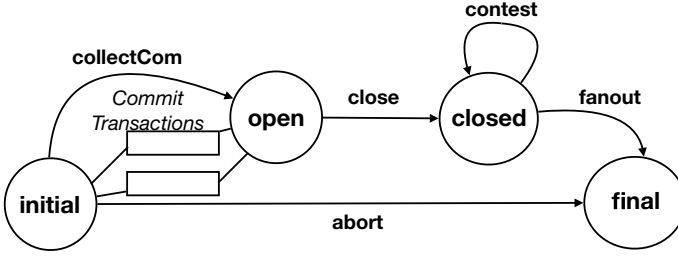


Fig. 1 The simplified Hydra head state machine includes four states *initial*, *open*, *closed*, *final*. The transition from the initial to the open state requires parties to commit EUTxOs to the head via dedicated transactions

open, *closed*, *final*. The CEM starts in the *initial* state representing the intention to create a Hydra head. If not aborted, all parties can commit a set of EUTxO which are moved into the Hydra head when transitioning into the *open* state where the Hydra head is considered created and the parties can perform the consensus protocol to modify η_0 . At any time the head can move to the *closed* state by providing the latest version of the snapshot η_i . This can be disputed through the *contest* transition and provision of a more recent set η_j where $i, j \in \{0, \dots, n\}$, $j > i$. Upon conclusion of the dispute the latest set η_n is committed to the ledger.

3 Overview

This section first introduces the setting and the concepts relevant to this work. Then, we present the Interhead State Machine. Due to space constraints, we leave details on the implementation in form of a CEM to Appendix C.

Terminology. We adjust our terminology to be in line with the Hydra protocol. In the following we refer to a Hydra head H_b , $b \in \{0, 1\}$ and the respectively other head as H_{1-b} . The Interhead is denoted with H^\vee .

3.1 The General Setting

We assume two multi-party state channels H_0 and H_1 that are able to execute state machines. Any party can enforce the state and resume execution of the state machine on the ledger after duration T^{max} at latest. Each of both channels H_b , $b \in \{0, 1\}$ is executed between parties $H_b.\text{Parties} := \{\mathcal{P}_{b,0}, \dots, \mathcal{P}_{b,n}\}$, $n \in \mathbb{N}$. The aim of our work is to create a virtual channel H^\vee where parties $G_1 \subseteq (H_0.\text{Parties} \cap H_1.\text{Parties})$, $\text{Parties}^{int} \neq \emptyset$ are the group of intermediaries, and

parties $G_b \subseteq H_b$. Parties, $G_b \neq \emptyset$ are the participants of the virtual channel H^v originating from channel H_b respectively. The set of all participants is $G^v := G_0 \cup G_1$. The virtual channel contains $b^v = b_0 + b_1$, $b^v, b_0, b_1 \in \mathbb{N}$ coins, where b_b coins are locked within channel H_b respectively. Moreover, we assume that the head's enforceable state can be partitioned into two parts, e.g. the state is a set of EUTxO which is the case in Hydra.

The Communication Model and Time. We assume that communication between the parties happens through authenticated channels and is done within rounds such that a message sent at any round will be available to the recipients at the beginning of the following round. We assume there is a relation between a given communication round and the clock time [14–16] at which it is happening such that we use time and communication rounds interchangeably in the remainder.

The Adversarial Model. In line with related work, we assume a malicious adversary who can statically corrupt all but one, i.e. up to $n_0 + n_1 + n_i - 1$, parties at the beginning of the protocol. Upon corruption the internal state of a party is leaked to the adversary and all communication to and from the party goes through the adversary. The adversary can make any corrupted party deviate from the protocol arbitrarily. Moreover, the adversary can reorder messages and delay them until the following communication round.

3.2 The Approach

The construction consists of two parts, an Interhead State Machine and a protocol. An overview of our approach is depicted in Fig. 2. The state machine is what is implemented and potentially executed on the ledger, i.e. a smart-contract or a CEM. Nevertheless, in-line with existing offchain protocols, in the *optimistic* case where no party deviates from the protocol the state machine is executed within a head, i.e. no transactions are committed to the ledger. The aim of our construction is to extend the original Hydra state machine to be executed *across* two Hydra heads. However, in case of a dispute it can be opened on-chain to rely on the ledger's security properties as fallback.

Overview. We execute a state machine which is derived from the Hydra CEM in each head, each is one half of the full Interhead State Machine. These state machine halves are similar by having the same structure and are executed synchronously by the set of intermediaries G_i . The intermediaries commit a collateral to the state machine halves as an incentive to remain honest as well as an insurance to the Interhead participants G_0, G_1 that the Interhead can be opened on the ledger even if all intermediaries are malicious. While the state machine remains offchain if the parties collaborate, in case any intermediary acts maliciously and deviates from the protocol the Interhead is opened on the ledger. If at least one Intermediary is honest,

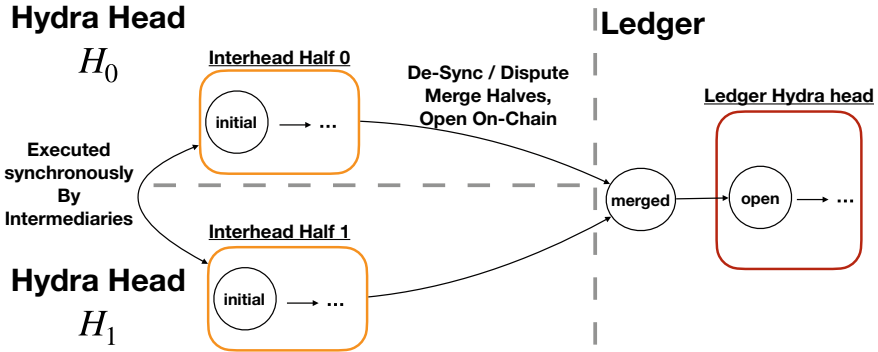


Fig. 2 The Interhead State Machine is split into two halves and executed within two Hydra heads. The halves have similar structure and are executed synchronously by a group of intermediaries, i.e. parties who participating in both heads. In case any party deviates from the protocol, e.g. by making both Interhead halves de-sync, the state machines are committed to the ledger, merged into one state machine and lastly transitioned to a regular head on the ledger

they will commit both Interhead halves to the ledger within time at most T^{max} , merge both halves into one state machine and transition that state machine into the state space of a regular Hydra head, thus opening the Interhead on the ledger. This releases the collateral of all intermediaries back to them. However, if all intermediaries are malicious, any honest party decommits their half of the Interhead to the ledger and transitions the half into a Hydra head by claiming the intermediaries’ collateral. Note that this bounds the amount of collateral that has to be paid by the Intermediaries, i.e. within a head H_b , the intermediaries have to commit collateral of at least as many coins as committed by the other head H_{1-b} .

3.3 Desired Properties and Challenges

The construction is designed to fulfill the following properties. Note that as soon as the Interhead State Machine transitions into the state space of the ledger channel, the security properties of the Hydra head hold.

Definition 1 (*Collateral Liveness*) If an intermediary is honest, their collateral is eventually available to them in an enforceable state.

Definition 2 (*EUTxO Liveness*) Eventually any honest party’s EUTxO within the virtual head’s enforceable state is available to them in an enforceable state outside the virtual head.

Definition 3 (*Balance Security*) The sum of a honest party’s coins is reduced only with their consent.

Security. The construction must be secure for all honest participants. Even if all other participants of the Interhead construction are behaving maliciously, the honest party cannot lose any coins. This requires that the Interhead construction fulfills **Balance Security**, **Collateral Liveness** and **EUTxO Liveness**.

Optimistic Offchain. Our construction must be optimistic offchain, i.e. if all parties collaborate, a virtual Hydra head can be constructed, used, and closed without commitment of any transactions to the ledger.

Bounded Collateral. The total collateral that is required to be collectively paid by the intermediaries is equal to the amount of coins b^v committed to the Interhead.

Unlimited Execution Time. The lifetime of the Interhead should not be bounded.

Multiple Intermediaries. Our construction allows for multiple intermediaries. While this does not provide any additional features to the construction itself, it is highly relevant for making the construction practical by allowing the required collateral to be committed by multiple parties. However, doing this securely is in itself a highly non-trivial challenge. Existing approaches for virtual channels [8, 9, 12] all have only one intermediary that is in a position to provide security to the construction, but in turn can be uniquely blamed if they deviate from the protocol. The challenge of multiple intermediaries is to ensure that the group of intermediaries is able to provide security to the construction the same way as with one intermediary. Nevertheless, honest intermediaries should not lose their collateral in case all remaining intermediaries behave maliciously.

4 The State Machine

Our approach is taking a Hydra state machine [4] and adapting concepts of virtual channels [8, 12, 13], notably General State Channel Networks [9], to create the Interhead State Machine that maintains a virtual Hydra head. Similar to Eltoo [21] we ensure that the virtual Hydra head can always be opened on the ledger as a regular head in case of dispute.

Time Phases and Assumptions. Execution of the state machine is structured into three phases, each operating under different assumptions and within disjunct time frames. (1) The *orderly* phase $T_O = [t_0, t_{C,\text{start}})$ assumes that at least one intermediary is honest and all parties collaborate. Note that while the Interhead's lifetime is bound by T_O , we provide facilities to extend it potentially arbitrarily. (2) The *conversion* phase $T_C = [t_{C,\text{start}}, t_{C,\text{end}})$ assumes that at least one intermediary is honest. We require that this phase's duration is at least $\max(H_b.T^{\max}, H_{1-b}.T^{\max}) + 2\Delta < t_{C,\text{end}} - t_{C,\text{start}}$. (3) The *punish* phase starts at $t_{C,\text{end}}$ and is indefinite. It assumes that at least one party is honest which is ensured due to the adversarial model. If any of

the assumptions during a phase is violated, the CEM escalates to the next phase by passage of time which prevents that any malicious parties can pause the execution of the CEM indefinitely.

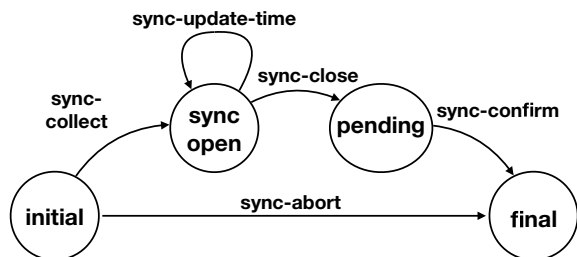
Setup. Each group G_0, G_1, G^v, G_i setups a multi-signature scheme as in Sect. 2 creating aggregate signature keys $K_{agg,0}, K_{agg,1}, K_{agg}, K_{agg,i}$ respectively.

4.1 The Interhead State Machine

The Interhead State Machine is structured using six states, namely `initial`, `sync_open`, `pending`, `final`, `merged` and `punished`. Here `initial` and `final` are the initial and the final states respectively. The `sync_open` state describes when the virtual head is open. The `merged` and `punished` are used to convert the Interhead into a regular Hydra head on the ledger, within the conversion phase and the punish phases. The `pending` state is used to detect and resolve de-synchronization. The Interhead State Machine contains the states, input symbols, state transitions, and final states of the Hydra state machine as it performs a state transition into the `open` state of the Hydra state space in case of dispute. For simplicity we abstract away from this and focus on the unique parts of the Interhead State Machine.

First, the parties in each head setup their half of the Interhead State Machine, starting in the `Initial` states respectively. The transitions that can be performed within a phase are structured according to the purpose of that phase displayed in Figs. 3–5. Both halves are operated in parallel, i.e. each `Initial` state spawns one thread of the overarching state machine. Intermediaries have to ensure that the initial states in both heads match such that in case of dispute the threads can be merged to be transitioned into the Hydra statespace. For this reason the `Initial` states contain data that (1) ensures the threads can be merged and (2) the threads cannot be merged with a different Interhead State Machine (3) the Hydra head maintains the same enforceable state as the Interhead. State transitions within the state machine are limited to time phases, i.e. their validity interval must fall entirely within one of the phases.

Fig. 3 State transitions in the orderly phase



Orderly Phase. Figure 3 illustrates states and transitions within the orderly phase. During this phase, the intermediaries have sole authority to perform state transitions by requiring a multi-signature corresponding to verification key $K_{agg,i}$. One instance of this partial state machine is executed within each head starting from the initial state. All transitions within this phase are executed synchronously, i.e. on both heads or on none denoted with input symbol prefix `sync`. This is ensured through a synchronization protocol with one caveat: A corrupted intermediary can attempt to de-sync both halves by acting in the last moment of the orderly phase and only providing their signature for one partial state machine. However, this can be detected by means of the `pending` state which acts as a buffer where intermediaries have to verify and confirm that no de-sync attempt occurred before the state machine can be closed. Detection of a de-sync attempt results in the state machine transitioning to the conversion phase.

The state machine can reach two states from the initial state: (1) The `sync-open` state is reached through the `sync-collect` input. This step collects a set of EUTxOs E_b from all participants, as well as a set of EUTxOs C_b from intermediaries that contain coins as collateral. The total amount of coins contained in C_b has to be at least as much as the total amount of coins contained in E_{1-b} . Upon reaching the `sync-open` state on both halves, the Interhead is opened and parties can modify its enforceable state. We omit the collection of EUTxO in Fig. 3 for simplicity. (2) The final state can be reached using the `sync-abort` command aborting execution and releasing all previously committed EUTxO.

The `pending` state can be reached from `sync-open` through the `sync-close` input. The parties G^V negotiate a partition of the Interhead's state into two sets of EUTxO, η_0, η_1 and the tuple $(final, \eta_b)$ is signed corresponding to the aggregate verification key $K_{agg,b}$. This *final* snapshot depends on the Hydra head to which it is submitted and contains the EUTxOs of its members as well as collateral from the intermediaries. Note that this step is similar to the concept of the optimistic head closure in Hydra [4]. The purpose of the `pending` state is to detect attempts of corrupted intermediaries to de-sync the partial state machines' executed on each head. Lastly the final state can be reached through the `sync-confirm` input from the `pending` state, releasing EUTxO according to the negotiated partitions. If the final state is reached within the orderly phase, the Interhead remains offchain and terminates. In the process, all EUTxOs previously committed by participants or committed as collateral by intermediaries are released. Otherwise, if any party stalls execution, does not collaborate or a de-sync attempt is detected, no further transitions occur in this phase and the state machine proceeds into the convert phase by passage of time.

Lastly, the duration of the orderly time phase can be extended by a transition from `sync-open` to itself while updating the timephase as stored within the state machine. Note that the conversion phase has a duration of $T_C = t_{C,end} - t_{C,start} \geq T^{max}$ and let $t_\delta = T_C - T^{max} \geq 0$. Then, we can extend the orderly timeframe by moving start and end of the conversion phase by t_δ , i.e. the new conversion phase will be at time $[t_{C,start} + t_\delta, t_{C,end} + \delta]$. This requires signatures from all parties, i.e. two aggregate signatures corresponding to K_{agg} and $K_{agg,i}$.

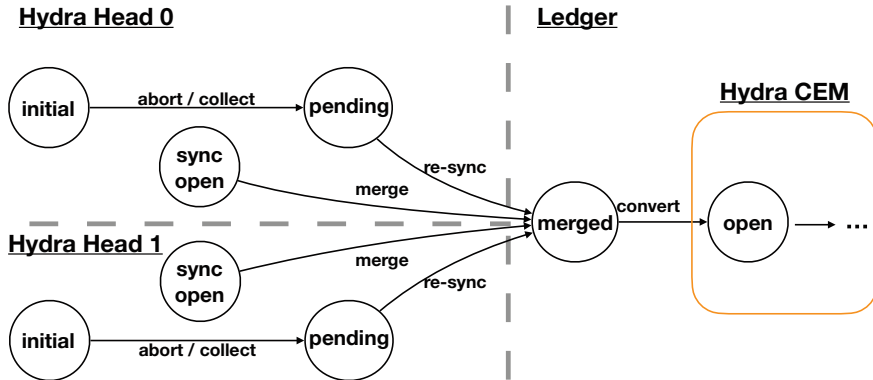


Fig. 4 State transitions in the conversion phase

Conversion Phase. This phase is illustrated in Fig. 4. Similar to the previous phase, all the intermediaries have sole authority to perform state transitions, however, now state transitions can be performed by any intermediary alone. In this phase, if there is a honest intermediary, both Interhead halves are committed to the ledger to be resolved there. This can be done through an incremental decommit of the state machine—in the case of Hydra heads—or by closing H_0 and H_1 . This requires time of up to T^{\max} . Then at first, if a state machine half is in the **initial** state it proceeds to the **pending** state. Next, the **merged** state can be reached in the following two cases: (1) If both halves are in the **sync-open** state and (2) if one half is in the **pending** state and the other in the **sync-open** state, i.e. a de-sync occurred. Note that if both halves are in the **pending** state, then no further transition occurs during this phase since this implies that no de-sync happened and we wait for the punish phase to confirm abort or closure of the Interhead. Lastly, we transition from the **merged** state to the **open** state of the Hydra state machine through the **convert** input. Note that we remove any time restriction for the transition from the **merged** state to the **open** state.

Punish Phase. The last phase is illustrated in Fig. 5. It is open-ended and any state transition can be performed by any party. (1) If the state machine is still in the **initial** or **pending** state, the state machine can be safely aborted and transition into the final state. (2) Otherwise, the state machine transitions into the open state of the Hydra state space. From the **sync-open** state the state machine transitions into the punished state with the **punish** input. The punished state is similar to the **merged** state with one exception. The intermediaries collateral is not released. Instead, the collateral is used to provide enough coins to allow opening the channel on the ledger and funding its enforceable state. Lastly, the state machine can still transition from the **merged** state to open the Interhead as a channel on the ledger, in case this has not happened in the conversion phase.

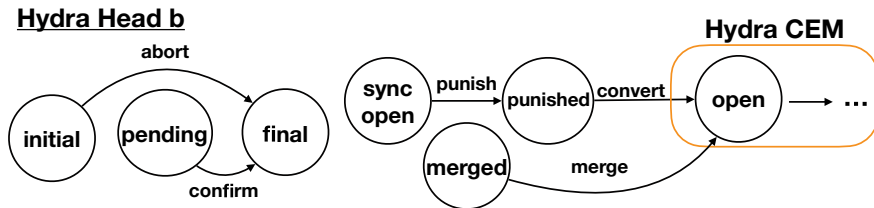


Fig. 5 State transitions in the punish phase

Iterative Construction. An Interhead maintains the same enforceable state as a Hydra head but it is setup across two enforceable states instead of one. Due to this an Interhead can be setup iteratively across any combination of two enforceable states. This allows for Interhead constructions across multiple hops of a network of Hydra heads. In the following, for simplicity we assume that the Interhead is created across two Hydra heads that were opened on the ledger.

Collateral and Fees. Let $b_i, i \in \{0, \dots, n_i\}$ be the collateral paid by intermediary \mathcal{P}_i and $c_j, j \in \{0, \dots, n_0 + n_1\}$ the amount of coins committed by participant \mathcal{P}_j . Moreover, let $C = \sum_{l=0}^{n_i} b_l = \sum_{l=0}^{n_0+n_1} c_l$. First, we assume the lifetime of the Interhead is fixed. Collateral is locked for at most until the end of the conversion phase which is after time $t_{dur} = t_{C,end} - t_0$. We extend the transition from initial to open states to require that each participant pays a fee proportional to the amount of coins they commit, i.e. \mathcal{P}_j commits fee $f_j = f \cdot t_{dur} \cdot c_j$ where $f \in \mathbb{R}^+$ is a scalar negotiated by all parties during setup. Let $f_\Sigma = \sum_{j=0}^{n_0+n_1} f_j$ be the total fees paid. Upon release of the collateral intermediary \mathcal{P}_i receives an additional fee proportional to the amount of collateral they paid, i.e. $\frac{b_i}{C} f_\Sigma$. For extending the lifetime of the Interhead via the sync-update-time transition by time t_{add} , each party pays an additional fee relative to t_{add} , i.e. $f_{j,add} = f \cdot t_{add} \cdot c_j$.

5 The Protocols

We define the behaviour of honest parties in form of two protocols. Algorithm 1 is executed among the intermediaries to coordinate state transitions, while Algorithm 2 is executed by all parties to close the Interhead (Figs. 6 and 7).

Setup. At the beginning, the parameter of the initial state are negotiated between all participants and intermediaries, and the intermediaries need to verify that both initial states match and can be merged within the conversion phase. The initial state contains aggregate signatures corresponding to both verification keys K_{agg} and $K_{agg,i}$. Any party provides their signature for this only in case of a positive verification of the relevant initial state.

Algorithm 1 Synchronization

```

1: function SYNC_TRANSITION( $tr_0, tr_1, i, aux$ )
2:   if  $\neg$  (VERIFY_TX( $tr_0, i, aux$ )  $\wedge$  VERIFY_TX( $tr_1, i, aux$ )) then return
3:   end if
4:    $\sigma_{agg} \leftarrow$  EXCHANGE_SIGNATURES( $i||aux, K_{agg,i}$ )
5:    $tr_0^s \leftarrow$  ADD_SIGNATURE( $tr_0, \sigma_{agg}$ ),  $tr_1^s \leftarrow$  ADD_SIGNATURE( $tr_1, \sigma_{agg}$ )
6:   COMMIT_VERIFY( $tr_0^s, H_0$ ), COMMIT_VERIFY( $tr_1^s, H_1$ )
7: end function

```

Fig. 6 Synchronous execution of a transition on both Hydra heads**Algorithm 2** Optimistic Closure

```

1: function OPTIMISTIC_CLOSURE( $\eta_0, \eta_1$ )
2:   if  $\neg$  (HAS_COLLATERAL( $\eta_0, \eta_1$ )  $\wedge$  CONSENT( $\eta_i, H_i$ )  $\wedge$  IS_PARTITION( $\eta^v, \eta_0, \eta_1$ )) then
3:     BROADCAST( $\perp, \eta_0, \eta_1$ ) return
4:   end if
5:   BROADCAST_EXCHANGE( $\eta_0, \eta_1, consent$ )
6:    $\sigma_{agg} \leftarrow$  EXCHANGE_SIGNATURES( $final||\eta_0||\eta_1, K_{agg}$ )
7:    $tr_0 \leftarrow$  CLOSE_TR( $\eta_0$ ),  $tr_1 \leftarrow$  CLOSE_TR( $\eta_1$ )
8:   SYNC_TRANSITION( $tr_0, tr_1, sync-close, \sigma_{agg}$ )
9: end function

```

Fig. 7 Protocol to close the Interhead offchain

Synchronization. Algorithm 1 describes synchronized execution of both partial CEMs. It is executed by each intermediary on the same input. No intermediary can execute multiple instances of the function concurrently which can be achieved by utilization of a mutex. The function's inputs are the two transactions tr_0, tr_1 that are to be executed on heads H_0, H_1 respectively, as well as input symbol i and auxiliary input aux of the transition that is performed. Then function VERIFY_TX checks that the transitions are valid except for the required aggregate signature of the intermediaries, and that they perform the correct state transition. If this is not the case, the function terminates. Otherwise EXCHANGE_SIGNATURE makes the intermediaries collaborate to complete the transactions by adding the required aggregate signature corresponding to their aggregate verification key $K_{agg,i}$. First each intermediary computes signature $\sigma = ms_sign(\Pi, sk, i||aux)$ and broadcasts it. Once an intermediary received the set S of all signatures they compute the aggregate signature $\sigma_{agg} = ms_agg_sign(\Pi, V, S, i||aux)$ and add it to the transitions. Lastly, COMMIT_VERIFY commits the completed transactions on their respective heads and only returns after they have been included in the head's snapshot.

Optimistic Closure. The protocol shown in Algorithm 2 takes two proposed snapshots η_0, η_1 and attempts to close the Interhead offchain. Similar to Algorithm 1 only one instance of the protocol can be executed concurrently enforceable through a mutex. The protocol is executed by intermediaries and participants. First, IS_PARTITION checks that η_0 and η_1 are a partition of η^v except for collateral, HAS_COLLATERAL checks whether the snapshots together have all of the party's collateral if applicable

and **CONSENT** verifies that the party agrees to close the Interhead with EUTxOs in η_0 and η_1 going to heads H_0, H_1 respectively. If any check fails, the party broadcasts symbol \perp to all parties to indicate that execution failed and terminates. Concurrent to execution of the protocol, parties listen towards receipt of \perp upon which they terminate the protocol as well. Otherwise, the parties communicate and exchange consent of the Interhead's closure. If a consent message from all parties was received, the parties collaborate to create the required transactions, including the aggregate signature of $\text{final}||\eta_0||\eta_1$ as auxiliary input. The intermediaries execute closure of the Interhead through protocol **SYNC_TRANSITION** as described in Algorithm 1.

6 Analysis

Collateral and Fees. The intermediaries experience a cost of opportunity by locking up collateral. A metric to estimate this cost [13, 19] is the product of coins locked $b \in \mathbb{N}$ and the duration they are locked $t \in \mathbb{N}$. The participants pay a fee relative to this to incentivize and compensate the intermediaries. Remember the mechanism of payment of fees as described in Sect. 4.1. We denote the total amount of collateral C which is locked away for time t_{dur} and a scalar f that describes the amount of fees paid per collateral and time. The total amount of fees paid is chosen to be relative to the above metric of cost of opportunity $o = bt = C \cdot t_{\text{dur}}$: The total amount of fees paid is $f_{\sigma} = \sum_{i=0}^{n_0+n_1} f_i = \sum_{i=0}^{n_0+n_1} f \cdot t_{\text{dur}} \cdot c_i = f \cdot t_{\text{dur}} \sum_{i=0}^{n_0+n_1} c_i = f \cdot t_{\text{dur}} \cdot C = f \cdot o$.

Novel to our setting is that there can be multiple intermediaries which can commit different amounts of collateral individually. The fees that are paid out to each intermediary are relative to their individual collateral respectively. In addition, we opt to have participants pay a fee relative to the number of coins they commit into the Interhead. This makes the rate of fees paid per coin committed constant. In comparison, if all parties pay an equal fee the rate of fees paid per committed coin can vary significantly making it prohibitively expensive for parties that commit a relatively low number of coins. However, this approach has a drawback, namely updating the amount of coins held by each party for sake of computing fees dynamically requires updating the Interhead's snapshot including resolving of disputes. Moreover, coins that are locked in a CEM that is executed within the Interhead might not be uniquely attributed to any party which requires additional negotiation of fees between the CEMs participants.

6.1 Security

We provide security statements for the Interhead construction by arguing through analysis of the possible paths a run through the state machine can take.

Lemma 1 *A honest intermediary can prevent that a run through the Interhead CEM reaches any **punished** state.*

Proof If all parties collaborate the state machine terminates through optimistic closure. Such a run does not include the **punished** state. Otherwise, both halves are committed to be resolved on the ledger by the honest intermediary and either (1) the Interhead reaches the conversion phase with both interhead halves in the same state or (2) a de-sync attempt happened. Case (1) can only occur if both heads are in the initial state, the **pending** state or the **sync-open** state. In the first case, both halves transition into the **pending** state as in the second case. From there the state machine reaches the final state and terminates during the punish phase without reaching the **punished** state. In the third case, both state machine halves will be merged by the honest intermediary without reaching the **punished** state. Lastly, (2) if a de-sync attempt occurred during a **sync-update-time** transition both halves are in the **sync-open** state but with different conversion phases. However, we require that even if both conversion phases are not equal, they overlap for at least the minimum duration of the conversion phase giving the honest intermediary enough time to perform the merge transition on-chain. Lastly, if one half is in the **sync-open** state and the other in the **pending** state, the intermediary can merge both halves as above. This is even the case if the **pending** state is reached through the initial state: Due to the synchronization protocol and design of the Interhead it is required that if the **sync-open** state on one half is reached then both halves can move into the **sync-open** state as all inputs, i.e. EUTxO and collateral, must be available. Therefore all EUTxO and collateral is available to transition into the **merged** state as well. In neither case a run contains the **punished** state.

Lemma 2 (Termination) *If at least one party is honest and attempts to terminate the head, the Interhead CEM eventually reaches a final state or transitions into the Hydra CEM state space.*

Proof First, if all parties collaborate and do not behave maliciously, the Interhead CEM can reach a final state across two paths during the orderly phase, i.e. either through an abort, or through optimistic head closure. Otherwise, as the honest party wishes to terminate the head, they will not collaborate to extend the duration of the orderly phase. Then the state machine will transition into the conversion phase by passage of time. If the party is an intermediary they can perform a transition into the **merged** state, or if they are not they wait for the conversion phase to end to perform a transition to the **punished** phase. From both states a transition into the Hydra CEM state space exist.

Theorem 1 (Collateral Liveness) *If at least one intermediary is honest, the Interhead construction has the collateral liveness property.*

Proof We note that there are two states that enforce that any collateral, that was committed prior, is made available within any enforceable state. These two states are the final state and the **merged** state. As shown in Lemma 2, eventually the Interhead

CEM either reaches a final state or the Hydra CEM state space. There is only one path through the Interhead CEM a run can take that does not end in a final state or contains the **merged** state. This is when the Hydra CEM has a run that contains the **punished** state. However, as shown in Lemma 1 any honest intermediary can avoid a run containing this state.

Theorem 2 (EUTxO Liveness) *If at least one party is honest, the Interhead construction has the EUTxO liveness property.*

Proof Due to Lemma 2 we have two cases to consider for any honest party \mathcal{P} . Either the CEM reaches a final state, or it reaches the Hydra CEM state space. In the latter case, we are finished. In the former case we have two cases. For one, the CEM can abort which unlocks all previously committed EUTxO in which case we are finished. Otherwise, the final state is reached through the **sync-open** and **pending** states through an optimistic closure. In that case, the EUTxO that are unlocked depend on the negotiation during the optimistic closure protocol. \mathcal{P} 's collaboration of an aggregate multisignature is required to perform optimistic closure. If \mathcal{P} is member in Head H_b it verifies that all of the EUTxO it is related with are present in the snapshot partition η_b or in either partition if it is member of both heads. In either case, all EUTxO within the partitions are made available within the Hydra heads \mathcal{P} is member of.

Theorem 3 (Balance Security) *If at least one party is honest, the Interhead construction has the balance security property.*

Proof This follows directly from Theorems 1 and 2.

7 Conclusion

In this work we present the Interhead construction, an approach to create virtual Hydra heads that goes beyond simple payments but instead allows for the execution of arbitrary state machines between participants across a network of Hydra heads. We define security properties **Collateral Liveness**, **EUTxO Liveness** and **Balance Security** and prove them in the presence of a malicious adversary. We present the first virtual channel construction that supports channels with an arbitrary number of parties and that collateral is contributed by multiple intermediaries. As future work we aim to generalize the construction beyond creating a virtual Hydra head as well as experimental evaluation.

A In-Depth Background

We provide an in-depth description of the Interhead CEM in Appendix C. In the following we provide additional background relevant to it (Fig. 8).

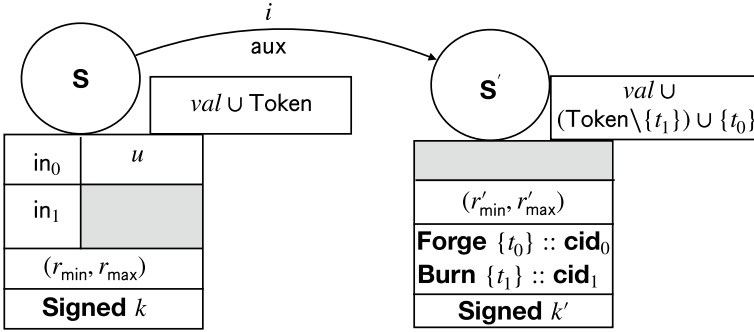


Fig. 8 Illustration of State Transition $S \rightarrow S'$ on input (i, aux) . The box below a state displays information on the transaction constraints for the transaction that performed the state transition. The box to the right of the state displays an overview of the *value* field of the EUTxO that represents the state. Transaction fields that are empty or implicit from context are grayed out or omitted for simplification

A.1 EUTxO_{MA}

The EUTxO model is extended by EUTxO_{MA} [3] to add multi-assets support. A EUTxO_{MA} is defined as a EUTxO but allows the *value* field to carry non-fungible tokens in addition to fungible coins. Moreover a transaction in the EUTxO_{MA} model has two more entries, i.e. it is of form $(I, O, \text{forge}, \text{fpss}, S)$ where **forge** is a token bundle that can define a positive amount of token in case they are minted, or a negative amount of token in case they are burned. Moreover, **fpss** is a forging policy script taking the validation context σ as input and evaluates whether the transaction including its **forge** field is admissible. In the remainder we assume EUTxO_{MA}, however we continue using the term EUTxO for brevity.

A.2 Thread Token

A design pattern that allows to enforce that a given CEM (1) started in a valid initial state and (2) is unique compared to other instances of similar CEMs is using thread token. The validator of an initial state requires creation of a thread token with respective forging policy script. The token will be kept in all EUTxO value fields through a run of the CEM until it reaches a final state in which it is forced to be burned.

B Hydra-Specific Concepts

B.1 General Purpose Token

Hydra heads are limited in that it is not possible to forge and burn arbitrary token. Special purpose token, such as thread token, can only be forged within a specific context specified within its forging policy script. A transaction forging such a token cannot be included in an enforceable state as this would result in the token being forged within a Hydra CEM state transition which would likely violate its forging policy script. A possible workaround is as follows: A generalized token is forged in an arbitrary context and as of such can be forged during any state transition of a Hydra head. Generalized token can be created either as fungible or non-fungible token. A CEM that makes use of token to perform functionality does not forge or burn the token, but instead takes the required amount of token as input when required, and releases the token in the CEM's final state at latest.

B.2 The Multi-Threaded CEM

We extend the notion of thread token by allowing CEMs to hold multiple thread token. A CEM can spawn threads to be executed in parallel by having a transaction contain multiple EUTxO in its outputs, each representing a separate CEM state and holding at least one thread token. In turn, multiple threads can be merged into a single thread by having a transaction spending multiple EUTxO representing CEM states, consuming their thread token and defining one EUTxO in its output that contains all thread token in its `value` field. We use multithreading in two cases. For one, we use multiple threads—one thread per identity—to efficiently collect EUTxO that are to be moved to the virtual head. Note that this is similar to how the Hydra CEM collects EUTxO [4] to be moved into a head. Second, we initially spawn one thread in each head, i.e. each instance of a Interhead CEM has exactly two initial states containing one thread token each. If the Interhead resolves optimistically, the CEM remains separate and the threads are never merged. However, in case of a dispute or a lack of collaboration, when the Interhead is converted into a regular Hydra head, the threads will be merged.

C The CEM Construction

In the following we provide details of the CEM by describing each state as well as the constraints provided by each verifier. With this implementation we explicitly assume that the protocol is executed within Hydra heads and moreover maintains a virtual Hydra head which is opened on the ledger in case of dispute. We illustrate

key parts of the CEM using Figs. 9 and 11. Moreover, due to space constraints we do not formally define each verifier's behaviour but describe it on a high level only.

C.1 Parameters

The parameters under which the Interhead CEM is executed are negotiated among all participants and intermediaries in the beginning. We structure the data stored within the CEMs state in data δ^v that is required for opening the virtual head on the ledger, data δ_c that is common to the two partial CEMs in both H_0 and H_1 as well as data δ_b that is only relevant in each individual head H_b .

First, δ^v is a tuple of form $(K_{agg}, \eta, h_{MT}, n, T, cid_0, cid_1)$ where the first four parameter are the virtual head's state and cid_b is a non-fungible token $t_{s,b}$ in head H_b which is used as a unique identifier for the CEM half. These parameter are not derived during execution of the CEM, but represent a commitment from the Interhead participants to create a virtual head with the respective parameters and token. For one, δ^v is used to ensure that always the same virtual head is created, even if only data from one partial CEM is available which is the case when the CEM enters the punish phase. The token $t_{s,b}$ is stored to ensure that the Interhead CEM instance is unique and that both partial CEMs are tied to another, i.e. no Interhead halve can be merged with another similar Interhead instance. Note that the EUTxO contained in η are not allowed to contain non-fungible token.

Second, δ_c consists of tuple $(K_{agg,i}, h_{MT,i}, n_i, T_o, T_c)$ which contains data on the intermediaries, i.e. their aggregate verification key $K_{agg,i}$, the head of the Merkle tree containing all individual verification keys $h_{MT,i}$ and the number of intermediaries n_i . Moreover it contains points in time $T_o, T_c \in \mathbb{N}$ specifying the end of the orderly and conversion time phases respectively. Verifier ensure that a transition happens within the orderly timezone by ensuring that for the transition's time frame $[r_{min}, r_{max}]$ holds that $r_{max} \leq T_o$. Similarly verifier ensure that a transaction is within the conversion time frame by checking that $T_0 < r_{min} < r_{max} \leq T_c$. Lastly a transition happens in the punish phase if $r_{min} > T_c$.

Lastly, δ_b consists of tuple $(b, K_{agg,b}, \eta_b, h_{MT,b}, n_b, col_b)$ where $b \in \{0, 1\}$ is a bit identifying the order of both partial CEMs, $K_{agg,b}$ is an aggregate verification key, η_b is a commitment of the EUTxO that the parties will move to the virtual head and it must hold that $\eta = \eta_0 \cup \eta_1$, $h_{MT,b}$ is the root of the Merkle tree consisting of the individual verification keys of G_b , $n_b = |G_b|$ is the number of participants joining from H_b , and col_b is the collateral required to be paid by the intermediaries. Note that col_b has to be at least as large as the amount of coins contained in the EUTxO in η_{1-b} . The quantity and type of fungible token submitted in the collateral has to match the number of token submitted by the participants.

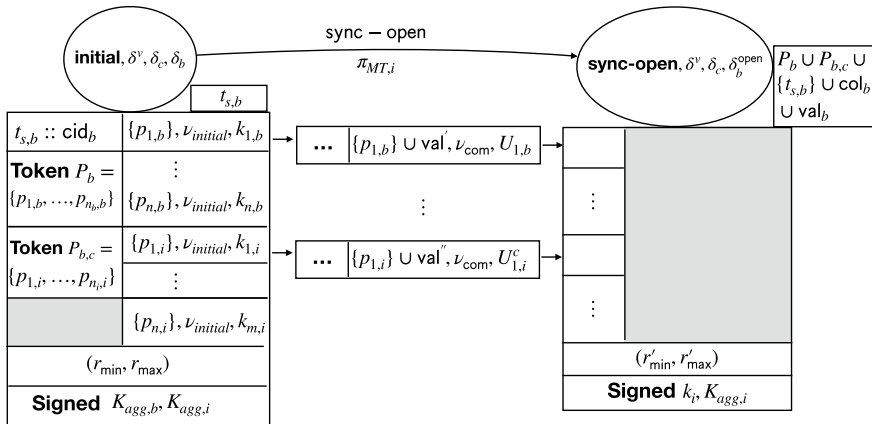


Fig. 9 Transition from initial to open state limited to the orderly phase

C.2 The Orderly Phase

The Initial State. The initial state is created in both heads H_b with parameter δ^v , δ_c , δ_b and is illustrated in Fig. 9. Each participant and intermediary verify that the parameter are as negotiated and, moreover, the intermediaries ensure that both initial states match and can be converted during the conversion phase in case of dispute. The transaction that sets up the initial state is signed using the aggregate verification keys of the participants in the respective head as well as the intermediaries. A party does only sign the transaction after positively verifying its correctness. The initial state requires that a non-fungible token as an ID, as well as a set of participation tokens [4] are provided. We require one participation token for each participant and each intermediary. These token ensure that each participant and each intermediary perform a commitment to the head and that all commitments can happen in parallel. The transition creates $n_b + n_i$ separate outputs, each containing one participation token and can be spent by exactly one participant and intermediary respectively.

Committing EUTxO. Similar to Hydra heads, all participants and intermediaries commit a set of EUTxO each to the Interhead which is done in parallel. Each participant and intermediary create one *commit* transaction that spends a participation token and several of their EUTxO within its inputs and stores information about them within its state $U_{i,b}$.

The Sync Open State. If all parties created a commit transaction, they can be spent by the *sync-open* transaction as shown in Fig. 9. The *sync-open* state verifies that the set of EUTxO $\eta = (U_{1,b}, \dots, U_{n_b,b})$ committed by the participants, matches their original commitment, i.e. $\eta = \delta_b \cdot \eta_b$. Moreover, it verifies that the collateral EUTxO $\eta_b^c = (U_{1,b}^c, \dots, U_{n_i,b}^c)$ that are committed by the intermediaries contain a sufficient

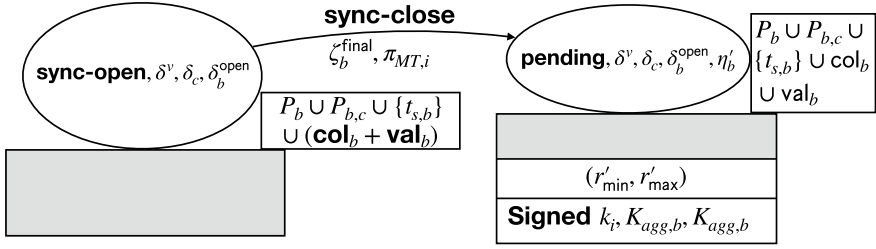


Fig. 10 First step of the optimistic closure during the orderly phase

number of coins and fungible token. We store δ_b^{open} in the CEMs state which equals δ_b but we replace $\delta_b.\text{col}_b$ with η'_b .

Aborting. Creation of the Interhead can be aborted by a transition from the initial state to the final state. The final state makes the EUTxO that were committed available on the ledger.

The Pending State. The pending state can be reached from the sync-open and is shown in Fig. 10. The pending state is used to give an opportunity for honest parties to either confirm head closure or to proceed to the conversion phase instead—in case a de-sync attempt was detected. The transition requires a *final*-annotated snapshot which transforms the EUTxO sets η_b and η_b^c into η'_b . The final snapshot η'_b contains two components. For one, it contains a partition of the EUTxO within the whole Interhead namely the EUTxO that will be made in head H_b upon closure. Moreover, it contains EUTxO that pay back the intermediaries' collateral. Note that the amount of coins that are paid back to the intermediaries within head H_b might be less than what was paid by the intermediaries upon opening the head, for instance, if participants from H_{1-b} performed payments to participants in head H_b . However, as the coins within the partial CEM is constant, the participants' coins will be taken from the coins submitted as collateral. However, in that case, this difference in coins will be available as additional collateral in H_{1-b} . Each intermediary has to ensure that the sum of collateral paid back to them in both heads is equal to the collateral they originally paid into creating the Interhead.

The Final State. In addition to aborting opening the Interhead, the final state can be reached from the pending state by a confirmation of the intermediaries. Similarly to the abort case, the UTxO sets are made available within the transaction's outputs. However, this time the EUTxO that are made available are taken from the final snapshot η'_b .

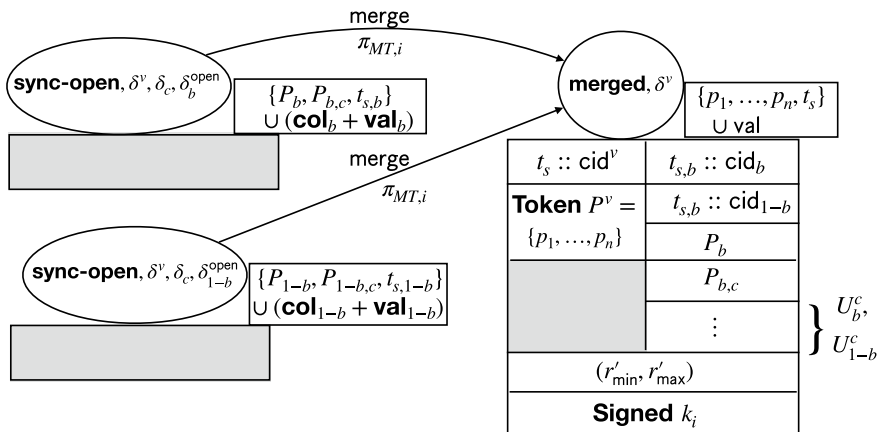


Fig. 11 Merging of both partial CEMs within the same enforceable state or the ledger. Transition can be performed from any combination of sync-open and pending states. Intermediaries' collateral is unlocked

C.3 The Conversion Phase

The conversion phase can conclude in two ways. For one, any intermediary can convert the Interhead CEM into a regular Hydra CEM. This requires that both partial CEMs are decommitted into a common enforceable state or the ledger respectively. This can happen by means of incremental decommits or closure of Hydra heads within time T^{max} . Note that no other state transitions are permitted within the Interhead CEM but conversion to a regular Hydra head. All transitions can be performed by an intermediary only, but now do not require a signature corresponding to the intermediaries' aggregate verification key $K_{\text{agg},i}$. For another, if the conversion has not been performed, in case no honest intermediary exists, the CEM transitions into the punish phase.

Abort. If there are any participants or intermediaries that have not yet committed EUTxO to the ledger, opening the Interhead can be aborted similarly to the way it is done during the orderly phase. However, if all participants and intermediaries committed EUTxO, aborting requires to transition into the pending state instead, because a de-sync attempt might have happened. Note that if the abort is permissible but the CEM transitioned into the pending state, it will be performed after during the punish phase.

The Merged State. Conversion to a regular CEM is by means of the merged state as shown in Fig. 11. Both partial CEMs can be merged, either from the open-sync state or the analogous pending state and any combination of both. The merge state requires that two generalized thread tokens are present and match $\delta^v.\text{cid}_0$ and $\delta^v.\text{cid}_1$. This ensures that only the intended partial CEMs can be merged as both non-fungible

thread token are unique. Moreover, it is verified that the EUTxO sets match the initial commitment, i.e. $\delta^v.\eta = \delta_0.\eta_0 \cup \delta_1.\eta_1$. This ensures that, in case all participants of one head as well as all intermediaries are corrupted, it is not possible for them to commit less EUTxO than required for the virtual Hydra head. The transaction releases all generalized token within its outputs, but similarly to the initial state it takes one thread token as well as $\delta^v.n$ participation token for all participants across both heads as input. Lastly, the CEM has one output for each set of EUTxO committed by the intermediaries to release their collateral in the same manner it is released in the final state. Note that from this point on, only information in δ^v is required and the remainder is removed from the state. Any participant can perform a transition into the *open* state of a regular Hydra CEM. We do not limit this transition to the conversion phase s.t. it can be performed indefinitely after start of the conversion phase. The state is directly derived from the data in δ^v , i.e. $K_{\text{agg}} = \delta^v.K_{\text{agg}}$, $\eta = \delta^v.\eta$, $h_{\text{MT}} = \delta^v.h_{\text{MT}}$, $n = \delta^v.n$, $T = \delta^v.T$.

C.4 The Punish Phase

The purpose of the last phase is to allow any honest party to open the virtual Hydra head between all participants, even in the case that all other parties are corrupted. The Interhead CEM transitions into a Hydra CEM without the need of merging both partial CEMs. To ensure that this is possible, the coins and fungible token that are necessary for opening the Hydra head are taken from the collateral of the intermediaries who will lose it in the process. All transitions in this phase can be performed by any participant or intermediary.

The Punished State. If the CEM is in the *sync-open* state when entering the punish phase the CEM will transition into a regular Hydra CEM via the *punish* state. This conversion can be performed within the same enforceable state in which the partial CEM is executed and thus requires no incremental decommit and neither closure of the Hydra head. The transitions from and to the *punished* state are similar to those to and from the *merged* state with the exception that we only verify correctness of the data from the local Hydra head, i.e. $\delta_b.\eta$ and $\delta^v.cid$. We re-use the existing thread token $t_{s,b}$ for the Hydra CEM. The collateral of the intermediaries is not made available through the transaction's outputs but used to finance conversion to the *open* state.

Open Ends. If the Partial CEM is in the *initial* state it can be aborted with a transition to the *final* state. Moreover, if the CEM was in the *pending* state it can now safely transition to the *final* state as no de-sync occurred. Lastly, the transition from the *merged* state to the *open* state of a Hydra CEM is open ended and thus can be performed in the punish phase as well.

References

1. Canetti, R. (2004). Universally composable signature, certification, and authentication. In *Proceedings. 17th IEEE Computer Security Foundations Workshop* (pp. 219–233). IEEE.
2. Chakravarty, M. M., Chapman, J., MacKenzie, K., Melkonian, O., Jones, M. P., & Wadler, P. (2020). The extended utxo model. In *4th Workshop on Trusted Smart Contracts*.
3. Chakravarty, M. M., Chapman, J., MacKenzie, K., Melkonian, O., Müller, J., Jones, M. P., Vinogradova, P., & Wadler, P. (2020). Native custom tokens in the extended utxo model. In *International Symposium on Leveraging Applications of Formal Methods* (pp. 89–111). Springer.
4. Chakravarty, M. M., Coretti, S., Fitzgi, M., Gazi, P., Kant, P., Kiayias, A., & Russell, A. (2021). Hydra: Fast isomorphic state channels. In *International Conference on Financial Cryptography and Data Security*. Springer.
5. Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sircer, E. G., et al. (2016). On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security* (pp. 106–125). Springer.
6. Decker, C., & Wattenhofer, R. (2015). A fast and scalable payment network with bitcoin duplex micropayment channels. In *Symposium on Self-Stabilizing Systems* (pp. 3–18). Springer.
7. Dziembowski, S., Eceky, L., Faust, S., Hesse, J., & Hostáková, K. (2019). Multi-party virtual state channels. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 625–656). Springer.
8. Dziembowski, S., Eceky, L., Faust, S., & Malinowski, D. (2017). *Perun: Virtual payment hubs over cryptocurrencies*. In Perun: Virtual Payment Hubs over Cryptocurrencies. IEEE.
9. Dziembowski, S., Faust, S., & Hostáková, K. (2018). General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (pp. 949–966). ACM.
10. EthHub. (2021). Sidechains. <https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/sidechains/>.
11. Itakura, K., & Nakamura, K. (1983). A public-key cryptosystem suitable for digital multisignatures. *NEC Research & Development*, 71, 1–8.
12. Jourenko, M., Larangeira, M., & Tanaka, K. (2020). Lightweight virtual payment channels. Cryptology ePrint Archive, Report 2020/998. <https://eprint.iacr.org/2020/998>.
13. Jourenko, M., Larangeira, M., & Tanaka, K. (2021). Payment trees: Low collateral payments for payment channel networks. In *International Conference on Financial Cryptography and Data Security*. Springer.
14. Katz, J., Maurer, U., Tackmann, B., & Zikas, V. (2013). Universally composable synchronous computation. In *Theory of Cryptography Conference* (pp. 477–498). Springer.
15. Kiayias, A., & Litos, O. S. T. (2019). A composable security treatment of the lightning network. *IACR Cryptology ePrint Archive*, 2019, 778.
16. Kiayias, A., Zhou, H. S., & Zikas, V. (2016). Fair and robust multi-party computation using a global transaction ledger. In M. Fischlin & J. S. Coron (Eds.), *Advances in Cryptology-EUROCRYPT 2016* (pp. 705–734). Berlin Heidelberg, Berlin, Heidelberg: Springer.
17. Mealy, G. H. (1955). A method for synthesizing sequential circuits. *The Bell System Technical Journal*, 34(5), 1045–1079.
18. Micali, S., Ohta, K., & Reyzin, L. (2001). Accountable-subgroup multisignatures. In *Proceedings of the 8th ACM Conference on Computer and Communications Security* (pp. 245–254).
19. Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., & McCorry, P. (2019). Sprites and state channels: Payment networks that go faster than lightning. In I. Goldberg, T. Moore (Eds.), *FC 2019. LNCS* (vol. 11598, pp. 508–526). Heidelberg: Springer. https://doi.org/10.1007/978-3-030-32101-7_30.
20. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.
21. PDecker, C., Russel, R., & Osuntokun, O. (2017). eltoo: A simple layer2 protocol for bitcoin. See <https://blockstream.com/eltoo.pdf>.

22. Poon, J., & Dryja, T. (2016). The bitcoin lightning network: Scalable off-chain instant payments. See <https://lightning.network/lightning-network-paper.pdf>.
23. Richards, S., & Wackerow, P. (2021). Plasma. <https://ethereum.org/en/developers/docs/scaling/plasma/>.
24. Richards, S., & Wackerow, P. (2021). Sidechains. <https://ethereum.org/en/developers/docs/scaling/sidechains/>.

Prediction Markets, Automated Market Makers, and Decentralized Finance (DeFi)



Yongge Wang 

Abstract This paper compares mathematical models for automated market makers (AMM) including logarithmic market scoring rule (LMSR), liquidity sensitive LMSR (LS-LMSR), constant product/mean/sum, and others. It is shown that though LMSR may not be a good model for Decentralized Finance (DeFi) applications, LS-LMSR has several advantages over constant product/mean based AMMs. This paper proposes and analyzes constant ellipse based cost functions for AMMs. The proposed cost functions are computationally efficient (only requires multiplication and square root calculation) and have certain advantages over widely deployed constant product cost functions. For example, the proposed market makers are more robust against slippage based front running attacks. In addition to the theoretical advantages of constant ellipse based cost functions, our implementation shows that if the model is used as a cryptographic property swap tool over Ethereum blockchain, it saves up to 46.88% gas cost against Uniswap V2 and saves up to 184.29% gas cost against Uniswap V3 which has been launched in April 2021. The source codes related to this paper are available at <https://github.com/coinswapapp> and the prototype of the proposed AMM is available at <http://coinswapapp.io/>.

Keywords Decentralized finance · Market scoring rules · Constant ellipse

1 Introduction

Decentralized finance (DeFi or open finance) is implemented through smart contracts (DApps) which are stored on a public distributed ledger (such as a blockchain) and can be activated to automate execution of financial instruments and digital assets. The immutable property of blockchains guarantees that these DApps are also tamper-proof and the content could be publicly audited.

Y. Wang (✉)
UNC Charlotte, Charlotte, NC 28223, USA
e-mail: yonwang@uncc.edu

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes
in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_12

213

DeFi applications range from automated markets (e.g., Uniswap [11] and Curve Finance), price oracles (e.g., Chainlink), to financial derivatives and many others. Most DeFi applications (e.g., Bancor [6] and Compound [7]) enable smart token transaction instantly by using price equilibrium mechanisms based on total availability supply (or called bonding curves), though still some of DeFi applications do not carry out instant transaction. In a blockchain system, traders submit their transactions to the entire blockchain network (e.g., stored in the mempool), a miner in the system collects these transactions, validates them, and puts them into a valid block that is eventually added to an immutable chain of blocks. These submitted transactions (e.g., the mempool for Ethereum could be viewed at <https://etherscan.io/txsPending>) are visible to all nodes. A malicious node (the miner itself could be malicious) may construct his/her own malicious transactions based on these observed transactions and insert her malicious transactions before or after the observed transactions by including appropriate gas costs (see, e.g., [14]). These malicious transactions take Miner Extractable Value (MEV) profit with minimal cost. With their own experience of failing to recover some tokens of 12K USD value in a Uniswap V2 pair (these tokens were recovered by a front running bot), Robinson and Konstantopoulos [10] describe the Ethereum blockchain as a Dark Forest. The flashbots website (<https://explore.flashbots.net/>) shows that the total extracted MEV by front running bots in the 24 hours of May 18, 2021 is around 8.6M USD. In addition to the front running attacks, it is also common to mount attacks against DeFi price oracles. In the DeFi market, a lender (a smart contract) normally queries an oracle to determine the fair market value (FMV) of the borrower's collateral.

This paper analyzes existing mathematical models for AMMs and discusses their applicability to blockchain based DeFi applications. One important consideration for the discussion is to compare the model resistance to front running attacks. Our analysis shows that though LS-LMSR is the best among existing models, it may not fit the blockchain DeFi application due to the following two reasons:

- LS-LMSR involves complicated computation and it is not gas-efficient for DeFi implementations.
- The cost function curve for LS-LMSR market is concave. In order to reflect the DeFi market principle of supply and demand, it is expected that the cost function curve should be convex.

Constant product based Uniswap AMM has been very successful as a DeFi swapping application. However, our analysis shows that Uniswap V2 [11] has a high slippage (in particular, at the two ends) and may not be a best choice for several applications. This paper proposes a constant ellipse based AMM model. It achieves the same model property as LS-LMSR but its cost function curve is convex and it is significantly gas-efficient for DeFi applications. At the same time, it reduces the sharp price fluctuation challenges by Uniswap V2. We have implemented and deployed a prototype CoinSwap based on our constant ellipse AMM during March 2021 (see <http://coinswapapp.io/>) and released a technical report [13] of this paper during September 2020. The CoinSwap has a controllable slippage mechanism and has a mechanisms for Initial Coin Offer (ICO). It should be noted that Uniswap

team was aware of their Uniswap V2 disadvantages that we have just mentioned and, independent of this paper, proposed the Uniswap V3 [12] (released during April 2021). Uniswap V3 tried to address this challenge by using a shifted constant product equation $(x + \alpha)(y + \beta) = K$. Though this shifted equation in Uniswap V3 resolves some of the challenges that constant ellipse AMM has addressed and it can implement some of the functionalities in CoinSwap (e.g., ICO and reduced slippage), it still does not have a smooth price fluctuation at two ends. Furthermore, the experimental data from CoinSwap project shows that Uniswap V3 has significant high gas costs than CoinSwap (to achieve the same functionality).

The structure of the paper is as follows. Section 2 gives an introduction to prediction markets and analyzes various models for automated market makers (AMM). Section 3 proposes a new constant ellipse market maker model. Section 4 compares various cost functions from aspects of the principle of supply and demand, coin liquidity, and token price fluctuation. Section 5 compares price amplitude for various cost functions and Sect. 6 discusses the implementation details.

2 Existing Models for Prediction Market Makers

2.1 Prediction Market and Market Makers

It is commonly believed that combined information/knowledge of all traders are incorporated into stock prices immediately (Fama [2] includes this as one of his “efficient market hypotheses”). For example, these information may be used by traders to hedge risks in financial markets such as stock and commodities future markets. With aggregated information from all sources, speculators who seek to “buy low and sell high” can take profit by predicting future prices from current prices and aggregated information. Inspired by these research, the concept of “information market” was introduced to investigate the common principles in information aggregation. Among various approaches to information market, a *prediction market* is an exchange-traded market for the purpose of eliciting aggregating beliefs over an unknown future outcome of a given event. As an example, in a horse race with n horses, one may purchase a security of the form “horse A beats horse B ”. This security pays off \$1 if horse A beats horse B and \$0 otherwise. Alternatively, one may purchase other securities such as “horse A finishes at a position in S ” where S is a subset of $\{1, \dots, n\}$. For the horse race event, the outcome space consists of the $n!$ possible permutations of the n horses.

For prediction markets with a huge outcome space, the continuous double-sided auction (where the market maker keeps an order book that tracks bids and asks) may fall victim of the *thin-market* problem. Firstly, in order to trade, traders need to coordinate on what or when they will trade. If there are significantly less participants than the size of the outcome space, the traders may only expect substantial trading activities in a small set of assets and many assets could not find trades at all. Thus the

market has a low to poor liquidity. Secondly, if a single participant knows something about an event while others know nothing about this information, this person may choose not to release this information at all or only release this information gradually. This could be justified as follows. If any release of this information (e.g., a trade based on this information) is a signal to other participants that results in belief revision discouraging trade, the person may choose not to release the information (e.g., not to make the trade at all). On the other hand, this person may also choose to leak the information into the market gradually over time to obtain a greater profit. The second challenge for the standard information market is due to the *irrational participation* problem where a rational participant may choose not to make any speculative trades with others (thus not to reveal his private information) after hedging his risks derived from his private information.

2.2 Logarithmic Market Scoring Rules (LMSR)

Market scoring rules are commonly used to overcome the thin market and the irrational participation problems discussed in the preceding section. Market scoring rule based automated market makers (AMM) implicitly/explicitly maintain prices for all assets at certain prices and are willing to trade on every assets. In recent years, Hanson's logarithmic market scoring rules (LMSR) AMM [4, 5] has become the de facto AMM mechanisms for prediction markets.

Let X be a random variable with a finite outcome space Ω . Let \mathbf{p} be a reported probability estimate for the random variable X . That is, $\sum_{\omega \in \Omega} \mathbf{p}(\omega) = 1$. In order to study rational behavior (decision) with fair fees, Good [3] defined a reward function with the logarithmic market scoring rule (LMSR) as follows:

$$\{s_{\omega}(\mathbf{p}) = b \ln(2 \cdot \mathbf{p}(\omega))\} \quad (1)$$

where $b > 0$ is a constant. A participant in the market may choose to change the current probability estimate \mathbf{p}_1 to a new estimate \mathbf{p}_2 . This participant will be rewarded $s_{\omega}(\mathbf{p}_2) - s_{\omega}(\mathbf{p}_1)$ if the outcome ω happens. Thus the participant would like to maximize his expected value (profit)

$$S(\mathbf{p}_1, \mathbf{p}_2) = \sum_{\omega \in \Omega} \mathbf{p}_2(\omega) (s_{\omega}(\mathbf{p}_2) - s_{\omega}(\mathbf{p}_1)) = b \sum_{\omega \in \Omega} \mathbf{p}_2(\omega) \ln \frac{\mathbf{p}_2(\omega)}{\mathbf{p}_1(\omega)} = bD(\mathbf{p}_2||\mathbf{p}_1) \quad (2)$$

by honestly reporting his believed probability estimate, where $D(\mathbf{p}_2||\mathbf{p}_1)$ is the relative entropy or Kullback Leibler distance between the two probabilities \mathbf{p}_2 and \mathbf{p}_1 . An LMSR market can be considered as a sequence of logarithmic scoring rules where the market maker (that is, the patron) pays the last participant and receives payment from the first participant.

Equivalently, an LMSR market can be interpreted as a market maker offering $|\Omega|$ securities where each security corresponds to an outcome and pays \$1 if the outcome is realized [4]. In particular, changing the market probability of $\omega \in \Omega$ to a value $\mathbf{p}(\omega)$ is equivalent to buying the security for ω until the market price of the security reaches $\mathbf{p}(\omega)$. As an example for the decentralized financial (DeFi) AMM on blockchains, assume that the market maker offers n categories of tokens. Let $\mathbf{q} = (q_1, \dots, q_n)$ where q_i represents the number of outstanding tokens for the token category i . The market maker keeps track of the cost function $C(\mathbf{q}) = b \ln \sum_{i=1}^n e^{q_i/b}$ and a price function for each token

$$P_i(\mathbf{q}) = \frac{\partial C(\mathbf{q})}{\partial q_i} = \frac{e^{q_i/b}}{\sum_{j=1}^n e^{q_j/b}} \quad (3)$$

It should be noted that the equation (3) is a generalized inverse of the scoring rule function (1). The cost function captures the amount of total assets wagered in the market where $C(\mathbf{q}_0)$ is the market maker's maximum subsidy to the market. The price function $P_i(\mathbf{q})$ gives the current cost of buying an infinitely small quantity of the category i token. If a trader wants to change the number of outstanding shares from \mathbf{q}_1 to \mathbf{q}_2 , the trader needs to pay the cost difference $C(\mathbf{q}_2) - C(\mathbf{q}_1)$.

Next we use an example to show how to design AMMs using LMSR. Assume that $b = 1$ and the patron sets up an automated market maker $\mathbf{q}_0 = (1000, 1000)$ by depositing 1000 coins of token A and 1000 coins of token B . The initial market cost is $C(\mathbf{q}_0) = \ln(e^{1000} + e^{1000}) = 1000.693147$. The instantaneous prices for a coin of tokens are $P_A(\mathbf{q}_0) = \frac{e^{1000}}{e^{1000} + e^{1000}} = 0.5$ and $P_B(\mathbf{q}_0) = \frac{e^{1000}}{e^{1000} + e^{1000}} = 0.5$. If this AMM is used as a price oracle, then one coin of token A equals $\frac{P_A(\mathbf{q}_0)}{P_B(\mathbf{q}_0)} = 1$ coin of token B . If a trader uses 0.689772 coins of token B to buy 5 coins of token A from market \mathbf{q}_0 , then the market moves to a state $\mathbf{q}_1 = (995, 1000.689772)$ with a total market cost $C(\mathbf{q}_1) = 1000.693147 = C(\mathbf{q}_0)$. The instantaneous prices for a coin of tokens in \mathbf{q}_1 are $P_A(\mathbf{q}_1) = 0.003368975243$ and $P_B(\mathbf{q}_1) = 295.8261646$. Now a trader can use 0.0033698 coins of token B to purchase 995 coins of token A from the AMM \mathbf{q}_1 with a resulting market maker state $\mathbf{q}_2 = (0, 1000.693147)$ and a total market cost $C(\mathbf{q}_2) = 1000.693147 = C(\mathbf{q}_0)$.

The above example shows that LMSR based AMM works well only when the outstanding shares of the tokens are evenly distributed (that is, close to 50/50). When the outstanding shares of the tokens are not evenly distributed, a trader can purchase all coins of the token with lesser outstanding shares and let the price ratio $\frac{P_A(\mathbf{q})}{P_B(\mathbf{q})}$ change to an arbitrary value with a negligible cost. This observation is further justified by the LMSR cost function curves in Fig. 1. The first plot is for the cost function $C(x, y, z) = 100$ with three tokens and the second plot is for the cost function $C(x, y) = 100$ with two tokens. The second plot shows that the price for each token fluctuates smoothly only in a tiny part (the upper-right corner) of the curve with evenly distributed token shares. Outside of this part, the tangent line becomes vertical or horizontal. That is, one can use a tiny amount of one token to purchase all outstanding coins of the other

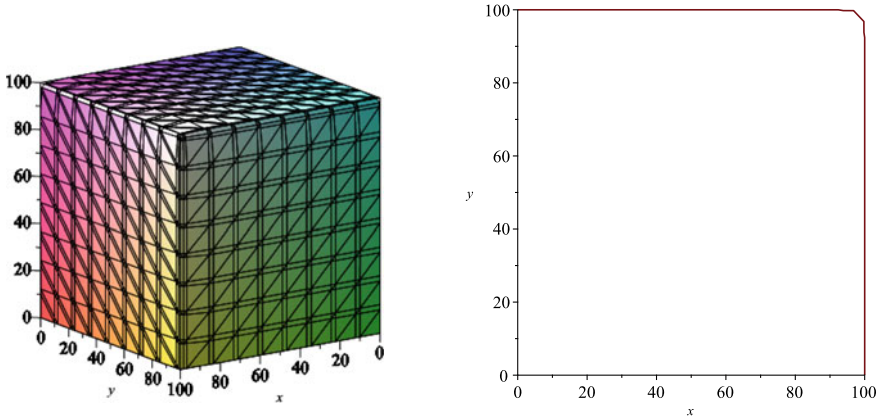


Fig. 1 LMSR market maker cost function curves for $C(x, y, z) = 100$ and $C(x, y) = 100$

token in the market maker. In a conclusion, LMSR based AMMs may not be a good solution for DeFi applications.

In the traditional prediction market, the three desired properties for a pricing rule to have include: *path independence*, *translation invariance*, and *liquidity sensitivity*. Path independence means that if the market moves from one state to another state, the payment/cost is independent of the paths that it moves. If path independence is not achieved, the adversary trader may place a series of transactions along a calculated path and obtain profit without any risk. Thus this is an essential property that needs to be satisfied. An AMM with a cost function generally achieves path independence. Thus all models that we will analyze in this paper (including our proposed constant ellipse AMM model) achieve path independence. On the other hand, the translation invariance guarantees that no trader can arbitrage the market maker without risk by taking on a guaranteed payout for less than the payout. As an example., a translation invariant pricing rule preserves the equality between the price of an event and the probability of that event occurring. Translation invariant rules also guarantee the “law of one price” which says that if two bets offer the same payouts in all states, they will have the same price. Liquid sensitivity property requires that a market maker should adjust the elasticity of their pricing response based on the volume of activity in the market. For example, as a generally marketing practice, this property requires that a fixed-size investment moves prices less in thick (liquid) markets than in thin (illiquid) markets. Though liquid sensitivity is a nice property to be achieved, a healthy market maker should not be too liquid sensitive (in our later examples, we show that Uniswap V2 is TOO liquid sensitive).

Definition 1 (see, e.g., Othman et al [9]) For a pricing rule P ,

1. P is path independent if the value of line integral (cost) between any two quantity vectors depends only on those quantity vectors, and not on the path between them.
2. P is translation invariant if $\sum_i P_i(\mathbf{q}) = 1$ for all valid market state \mathbf{q} .

3. P is liquidity insensitive if $P_i(\mathbf{q} + (\alpha, \dots, \alpha)) = P_i(\mathbf{q})$ for all valid market state \mathbf{q} and α . P is liquidity sensitive if it is not liquidity insensitive.

Othman et al [9] showed that no market maker can satisfy all three of the desired properties at the same time. Furthermore, Othman et al [9] showed that LMSR satisfies translation invariance and path independence though not liquidity sensitivity. In practice, the patron would prefer liquidity sensitivity instead of absolute translation invariance. By relaxing the translation invariance to $\sum_i P_i(\mathbf{q}) \geq 1$, Othman et al [9] proposed the Liquidity-Sensitive LMSR market. In particular, LS-LMSR changes the constant b in the LMSR formulas to $b(\mathbf{q}) = \alpha \sum_i q_i$ where α is a constant and requiring the cost function to always move forward in obligation space. Specifically, for $\mathbf{q} = (q_1, \dots, q_n)$, the market maker keeps track of the cost function $C(\mathbf{q}) = b(\mathbf{q}) \ln \sum_{i=1}^n e^{q_i/b(\mathbf{q})}$ and a price function for each token

$$P_i(\mathbf{q}) = \alpha \ln \left(\sum_{j=1}^n e^{q_j/b(\mathbf{q})} \right) + \frac{e^{q_i/b(\mathbf{q})} \sum_{j=1}^n q_j - \sum_{j=1}^n q_j e^{q_j/b(\mathbf{q})}}{\sum_{j=1}^n q_j \sum_{j=1}^n e^{q_j/b(\mathbf{q})}} \quad (4)$$

Furthermore, in order to always move forward in obligation space, we need to revise the cost that a trader should pay. In the proposed “no selling” approach, assume that the market is at state \mathbf{q}_1 and the trader tries to impose an obligation $\mathbf{q}_\delta = (q'_1, \dots, q'_n)$ to the market with $\bar{q}_\delta = \min_i q'_i < 0$. That is, the trader puts q'_i coins of token i to the market if $q'_i \geq 0$ and receives $-q'_i$ coins of token i from the market if $q'_i < 0$. Let $\bar{\mathbf{q}}_\delta = (-\bar{q}_\delta, \dots, -\bar{q}_\delta)$. Then the trader should pay $C(\mathbf{q} + \mathbf{q}_\delta + \bar{\mathbf{q}}_\delta) + \bar{q}_\delta - C(\mathbf{q})$ and the market moves to the new state $\mathbf{q} + \mathbf{q}_\delta + \bar{\mathbf{q}}_\delta$. In the proposed “covered short selling approach”, the market moves in the same way as LMSR market except that if the resulting market \mathbf{q}' contains a negative component, then the market \mathbf{q}' automatically adds a constant vector to itself so that all components are non-negative. In either of the above proposed approach, if $\mathbf{q} + \mathbf{q}_\delta$ contains negative components, extra shares are automatically mined and added to the market to avoid negative outstanding shares. This should be avoided in DeFi applications. In DeFi applications, one should require that \mathbf{q}_δ could be imposed to a market \mathbf{q}_0 only if there is no negative component in $\mathbf{q} + \mathbf{q}_\delta$ and the resulting market state is $\mathbf{q} + \mathbf{q}_\delta$. LS-LMSR is obviously path independent since it has a cost function. Othman et al [9] showed that LS-LMSR has the desired liquidity sensitive property. On the other hand, LS-LMSR satisfies the relaxed translation invariance $\sum_i P_i(\mathbf{q}) \geq 1$. This means that if a trader imposes an obligation and then sells it back to the market maker, the trader may end up with a net lost (this is similar to the markets we see in the real world). Figure 2 displays the curve of the cost function $C(x, y, z) = 100$ for LS-LMSR market maker with three tokens and the curve of the cost function $C(x, y) = 100$ for LS-LMSR market maker with two tokens. It is clear that these two curves are concave.

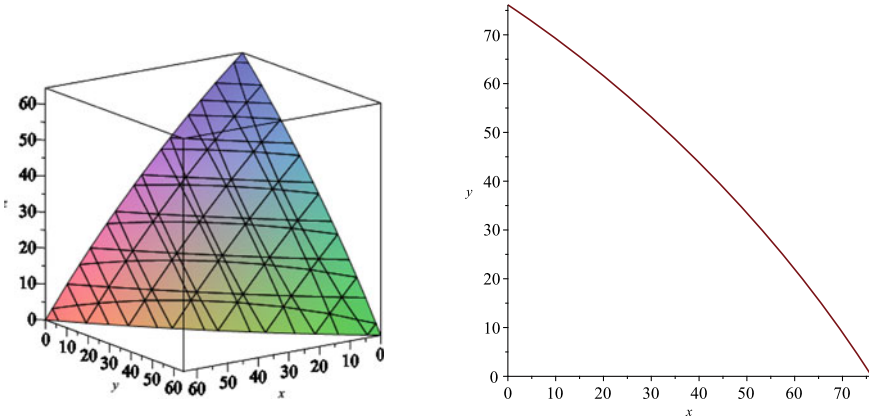


Fig. 2 LS-LMSR market maker cost function curves for $C(x, y, z) = 100$ and $C(x, y) = 100$

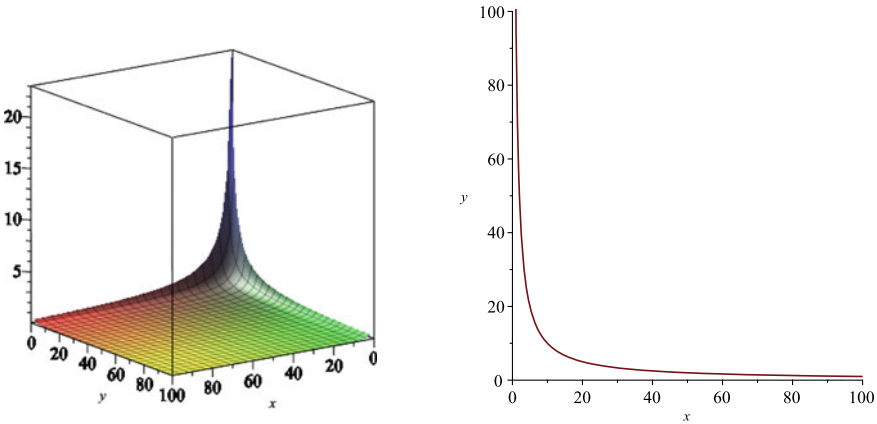


Fig. 3 Constant product cost function curves for $xyz = 100$ and $xy = 100$

2.3 Constant Product/Sum/Mean AMMs

Constant product market makers have been used in DeFi applications (e.g., Uniswap [11]) to enable on-chain exchanges of digital assets and on-chain-decentralized price oracles. In this market, one keeps track of the cost function $C(\mathbf{q}) = \prod_{i=1}^n q_i$ as a constant. For this market, the price function for each token is defined as $P_i(\mathbf{q}) = \frac{\partial C(\mathbf{q})}{\partial q_i} = \prod_{j \neq i} q_j$. Figure 3 shows the curve of the constant product cost function $xyz = 100$ with three tokens and the curve of the constant product cost function $xy = 100$ with two tokens.

The cost function $C(\mathbf{q}) = \prod_{i=1}^n q_i^{w_i}$ has been used to design constant mean AMMs [8] where w_i are positive real numbers. In the constant mean market, the price function for each token is $P_i(\mathbf{q}) = \frac{\partial C(\mathbf{q})}{\partial q_i} = w_i q_i^{w_i-1} \prod_{j \neq i} q_j$. Figure 4 shows the curve of the

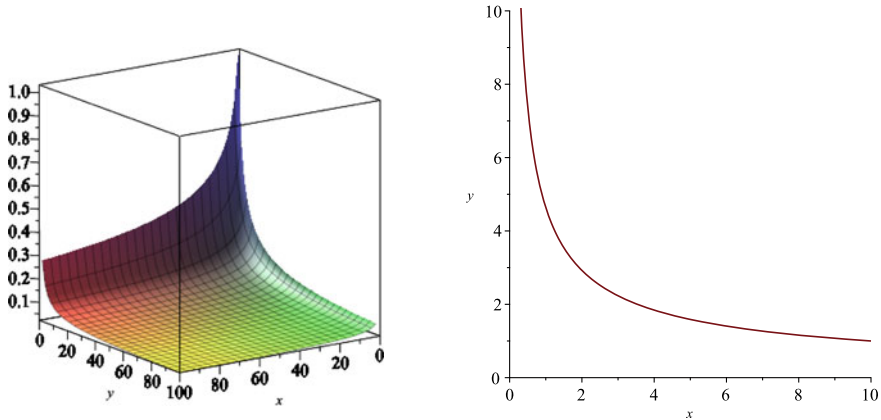


Fig. 4 Constant mean cost function curves for $xy^2z^3 = 100$ and $x^2y^3 = 100$

constant mean cost function $xy^2z^3 = 100$ with three tokens and the curve of the constant mean cost function $x^2y^3 = 100$ with two tokens.

One may also use the cost function $C(\mathbf{q}) = \sum_{i=1}^n q_i$ to design constant sum market makers. In this market, the price for each token is always 1. That is, one coin of a given token can be used to trade for one coin of another token at any time when supply lasts.

The curves in Figs. 3 and 4 show that constant product/mean/sum AMMs are highly liquidity sensitive when the distribution of the tokens are far from balanced market states (where the price fluctuates sharply). By the fact that there exist cost functions, constant product/mean/sum AMMs achieve path independence. It is also straightforward to check that constant product/mean AMMs are liquidity sensitive. By the fact (see [9]) that no market maker can satisfy all three of the desired properties at the same time, constant product/mean AMMs are not translation invariant. It is also straightforward to check that the constant sum AMM is liquidity insensitive. Since liquidity sensitivity is one of the essential market rules to be satisfied, in the remaining part of this paper, we will no long discuss constant sum models.

3 Constant Ellipse AMMs

Section 4 compares the advantages and disadvantages of LMSR, LS-LMSR, and constant product/mean/sum AMMs. The analysis shows that none of them is ideal for DeFi applications. In this section, we propose AMMs based on constant ellipse cost functions. That is, the AMM’s cost function is defined by

$$C(\mathbf{q}) = \sum_{i=1}^n (q_i - a)^2 + b \sum_{i \neq j} q_i q_j \quad (5)$$

where a, b are constants. The price function for each token is

$$P_i(\mathbf{q}) = \frac{\partial C(\mathbf{q})}{\partial q_i} = 2(q_i - a) + b \sum_{j \neq i} q_j.$$

For AMMs, we only use the first quadrant of the coordinate plane. By adjusting the parameters a, b in the equation (5), one may keep the cost function to be concave (that is, using the upper-left part of the ellipse) or to be convex (that is, using the lower-left part of the ellipse). By adjusting the absolute value of a , one may obtain various price amplitude and price fluctuation rates based on the principle of supply and demand for tokens. It is observed that constant ellipse AMM price functions are liquidity sensitive and path independent but not translation invariance. Figure 5 shows the curve of the constant ellipse cost function

$$(x - 10)^2 + (y - 10)^2 + (z - 10)^2 + 1.5(xy + xz + yz) = 350$$

with three tokens and the curve of the the constant ellipse cost function

$$(x - 10)^2 + (y - 10)^2 + 1.5xy = 121$$

with two tokens. As mentioned in the preceding paragraphs, one may use convex or concave part of the ellipse for the cost function. For example, in the second plot of Fig. 5, one may use the lower-left part in the first quadrant as a convex cost function or use the upper-right part in the first quadrant as a concave cost function. It is straightforward to verify that the constant ellipse AMM achieve path independence and liquidity sensitivity. Though constant ellipse AMM is not translation invariant, our analysis and examples provide evidence that in a constant ellipse AMM, a trader have certain risks for arbitraging the market maker on a payout for less than the payout (this is related to our analysis on the slippage in the later sections).

4 Supply-and-Demand, Liquid Sensitivity, and Price Fluctuation

Without loss of generality, this section considers AMMs consisting of two tokens: a USDT token where each USDT coin costs one US dollar and an imagined spade suit token ♠. The current market price of a ♠ token coin could have different values such as half a USDT coin, one USDT coin, two USDT coins, or others. In Decentralized Finance (DeFi) applications, the patron needs to provide liquidity by depositing coins of both tokens in the AMM. Without loss of generality, we assume that, at the time when the AMM is incorporated, the market price for a coin of spade suit token is

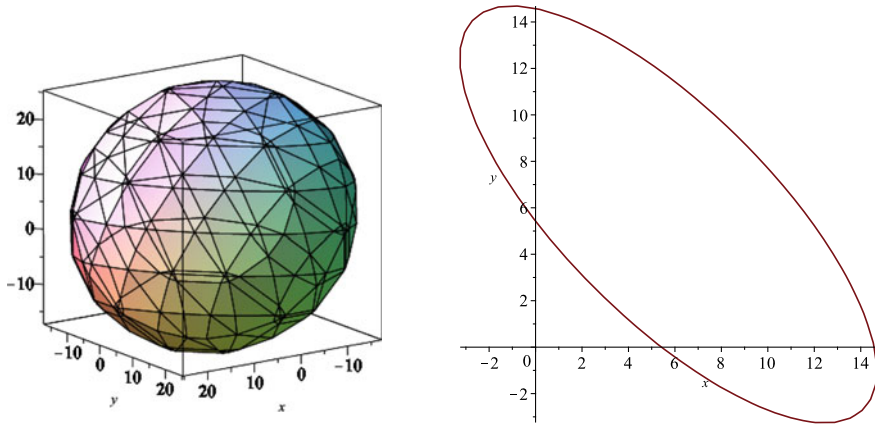


Fig. 5 Constant ellipse cost function curves for three and two tokens

equivalent to one USDT coin. For general cases that the market price for one ♠ coin is not equivalent to one USDT coin at the time when the market maker is incorporated, we can create virtual shares in the AMM by dividing or merging actual coins. That is, each share of USDT (respectively ♠) in the AMM consists of a multiple or a portion of USDT (respectively ♠) coins. One may find some examples in Sect. 5.

To simplify our notations, we will use $\mathbf{q} = (x, y)$ instead of $\mathbf{q} = (q_1, q_2)$ to represent the market state. In this section, we will only study the price fluctuation of the first token based on the principle of supply and demand and the trend of the price ratio $\frac{P_x(\mathbf{q})}{P_y(\mathbf{q})}$ which is strongly related liquid sensitivity. By symmetry of the cost functions, the price fluctuation of the second token and the ratio $\frac{P_y(\mathbf{q})}{P_x(\mathbf{q})}$ have the same property. In the following, we analyze the token price fluctuation for various AMM models with the initial market state $\mathbf{q}_0 = (1000, 1000)$. That is, the patron creates the AMM by depositing 1000 USDT coins and 1000 spade suit coins in the market. The analysis results are summarized in Table 1.

Table 1 Token price comparison

AMM type	Market cost	$P_x(\mathbf{q})/P_y(\mathbf{q})$	Tangent $\frac{\partial y}{\partial x}$
LS-LMSR	2386.29436	(0.648, 1.543)	(-1.543, -0.648)
Cons. product	1000000	$(0, \infty)$	$(-\infty, 0)$
Cons. sum	2000	1	-1
Cons. ellipse	50000000	(0.624, 1.604)	(-1.604, -0.624)

4.1 LS-LMSR

For the LS-LMSR based AMM, the market cost is

$$C(\mathbf{q}_0) = 2000 \cdot \ln(e^{1000/2000} + e^{1000/2000}) = 2386.294362.$$

At market state \mathbf{q}_0 , the instantaneous prices for a coin of tokens are $P_x(\mathbf{q}_0) = P_y(\mathbf{q}_0) = 1.193147181$. A trader may use 817.07452949 spade suit coins to purchase 1000 USDT coins with a resulting market state $\mathbf{q}_1 = (0, 1817.07452949)$ and a resulting market cost $C(\mathbf{q}_1) = 2386.294362$. At market state \mathbf{q}_1 , the instantaneous prices for a coin of tokens are $P_x(\mathbf{q}_1) = 0.8511445298$ and $P_y(\mathbf{q}_1) = 1.313261687$. Thus we have $P_x(\mathbf{q}_1)/P_y(\mathbf{q}_1) = 0.6481149479$. The tangent line slope of the cost function curve indicates the token price fluctuation stability in the automated market. The tangent line slope for the LS-LMSR cost function curve at the market state $\mathbf{q} = (x, y)$ is

$$\frac{\partial y}{\partial x} = -\frac{(x+y)\left(e^{\frac{x}{x+y}} + e^{\frac{y}{x+y}}\right)\ln\left(e^{\frac{x}{x+y}} + e^{\frac{y}{x+y}}\right) + y\left(e^{\frac{x}{x+y}} - e^{\frac{y}{x+y}}\right)}{(x+y)\left(e^{\frac{x}{x+y}} + e^{\frac{y}{x+y}}\right)\ln\left(e^{\frac{x}{x+y}} + e^{\frac{y}{x+y}}\right) + x\left(e^{\frac{y}{x+y}} - e^{\frac{x}{x+y}}\right)}.$$

For the LS-LMSR AMM with an initial state $\mathbf{q}_0 = (1000, 1000)$, the tangent line slope (see Fig. 6) changes smoothly and stays between -1.542936177 and -0.6481149479 . Thus the token price fluctuation is quite smooth. By the principle of supply and demand, it is expected that when the token supply increases, the token price decreases. That is, the cost function curve should be convex. However, the cost function curve for LS-LMSR market is concave. This can be considered as a disadvantage of LS-LMSR markets for certain DeFi applications. Though LS-LMSR does not satisfy the translation invariance property, it is shown in [9] that the sum of prices are bounded by $1 + \alpha n \ln n$. For the two token market with $\alpha = 1$, the sum of prices are bounded by $1 + 2 \ln 2 = 2.386294362$ and this value is achieved when $x = y$.

As an additional example of LS-LMSR AMMs, a trader may spend 10 USDT coins to purchase 10.020996 coins of spade suit token at market state \mathbf{q}_0 or spend 500 USDT coins to purchase 559.926783 coins of spade suit from the market state \mathbf{q}_0 with a resulting market state $(1500, 440.073217)$. Furthermore, in the market state $(1500, 440.073217)$, the value of one USDT coin is equivalent to the value of 1.260346709 coins of spade suit token.

4.2 Constant Product and Constant Mean

For the constant product AMM, the market cost is $C(\mathbf{q}_0) = 1000000$ and the constant product cost function is $x \cdot y = 1000000$. At market state \mathbf{q}_0 , the instantaneous token

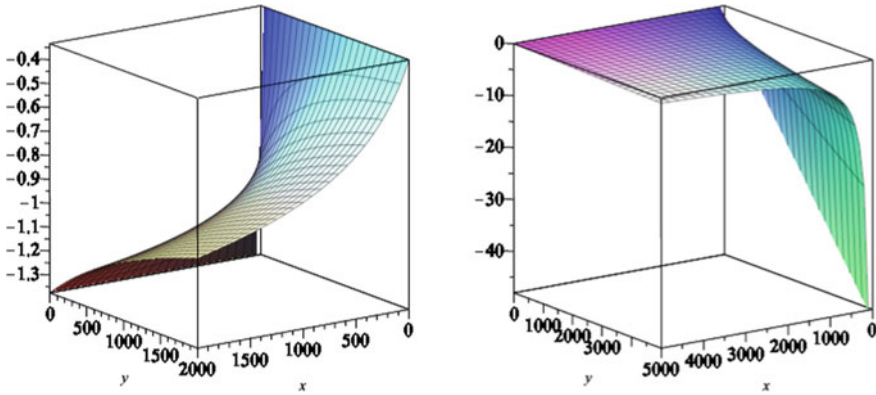


Fig. 6 Tangent line slopes for LS-LMSR (first) and constant product (second) cost functions

prices are $P_x(\mathbf{q}_0) = P_y(\mathbf{q}_0) = 1000$. Thus we have $\frac{P_x(\mathbf{q})}{P_y(\mathbf{q})} = 1$. A trader may use one USDT coin to buy approximately one coin of spade suit token and vice versa at the market state \mathbf{q}_0 . However, as market state moves on, the prices could change dramatically based on token supply in the market and the pool of a specific coin will never run out. Specifically, at market state \mathbf{q}_0 , a trader may spend 10 USDT coins to purchase 9.900990099 spade suit coins. On the other hand, a user may spend 500 USDT coins to purchase only 333.33333333 coins of spade suit token from the market state \mathbf{q}_0 with a resulting market state $\mathbf{q}_1 = (1500, 666.6666667)$. Note that in the example of LS-LMSR market example, at market state \mathbf{q}_0 , a trader can spend 500 USDT coins to purchase 559.926783 coins of spade suit. Furthermore, in the market state \mathbf{q}_1 , one USDT coin could purchase 0.4444444445 coins of spade suit token. The tangent line slope of the cost function curve at the market state $\mathbf{q} = (x, y)$ is

$$\frac{\partial y}{\partial x} = -\frac{P_x(\mathbf{q})}{P_y(\mathbf{q})} = -\frac{y}{x}.$$

That is, the tangent line slope for the cost function curve (see Fig. 6) can go from $-\infty$ to 0 and the token price fluctuation could be very sharp. Specifically, if the total cost of the initial market \mathbf{q}_0 is “small” (compared against attacker’s capability), then a trader/attacker could easily control and manipulate the market price of each coins in the AMM. In other words, this kind of market maker may not serve as a reliable price oracle. A good aspect of the constant product cost function is that the curve is convex. Thus when the token supply increases, the token price decreases. On the other hand, the sum of prices $P_x(\mathbf{q}) + P_y(\mathbf{q}) = x + y$ in constant product market is unbounded. Thus constant production cost function could not be used in prediction markets since it leaves a chance for a market maker to derive unlimited profit from transacting with traders.

For constant mean AMMs, Fig. 4 displays an instantiated constant mean cost function curve. The curve in Fig. 4 is very similar to the curve in Fig. 3 for the

constant product cost function. Thus constant mean AMM has similar properties as that for constant product AMM and we will not go into details.

4.3 Constant Ellipse

As we have mentioned in the preceding Sections, one may use the upper-right part of the curve for a concave cost function or use the lower-left part of the curve for a convex cost function. In order to conform to the principle of supply and demand, we analyze the convex cost functions based on constant ellipse. Constant ellipse share many similar properties though they have different characteristics. By adjusting corresponding parameters, one may obtain different cost function curves with different properties (e.g., different price fluctuation range, different tangent line slope range, etc). The approaches for analyzing these cost function curves are similar. Our following analysis uses the low-left convex part of the circle $(x - 6000)^2 + (y - 6000)^2 = 2 \times 5000^2$ as the constant cost function.

For AMMs based on this cost function $C(\mathbf{q}) = (x - 6000)^2 + (y - 6000)^2$, the market cost is $C(\mathbf{q}_0) = 50000000$. At market state \mathbf{q}_0 , the instantaneous prices for a coin of tokens are $P_x(\mathbf{q}_0) = P_y(\mathbf{q}_0) = -10000$. A trader may use 1258.342613 spade suit coins to purchase 1000 USDT coins with a resulting market state $\mathbf{q}_1 = (0, 2258.342613)$ and a resulting market cost $C(\mathbf{q}_1) = C(\mathbf{q}_0)$. At market state \mathbf{q}_1 , the instantaneous prices for a coin of tokens are $P_x(\mathbf{q}_1) = 12000$ and $P_y(\mathbf{q}_1) = 7483.314774$. Thus we have $\frac{P_x(\mathbf{q}_1)}{P_y(\mathbf{q}_1)} = 1.603567451$. The tangent line slope of the cost function curve at the market state $\mathbf{q} = (x, y)$ is

$$\frac{\partial y}{\partial x} = -\frac{P_x(\mathbf{q})}{P_y(\mathbf{q})} = -\frac{x - 6000}{y - 6000}.$$

This tangent line slope function (see Fig. 7) changes very smoothly and stays in the interval $[-1.603567451, -0.6236095645]$. Thus the token price fluctuation is quite smooth. Furthermore, this cost function has a convex curve which conforms to the principle of supply and demand. That is, token price increases when token supply decreases. For constant ellipse cost function market, the sum of prices are bounded by $P_x(\mathbf{q}) + P_y(\mathbf{q}) = 2(x + y) - 4a$. Similar bounds hold for constant ellipse cost function market. Thus, when it is used for prediction market, there is a limit on the profit that a market maker can derive from transacting with traders.

Figure 8 compares the cost function curves for different AMMs that we have discussed. These curves show that constant ellipse cost function is among the best ones for DeFi applications.

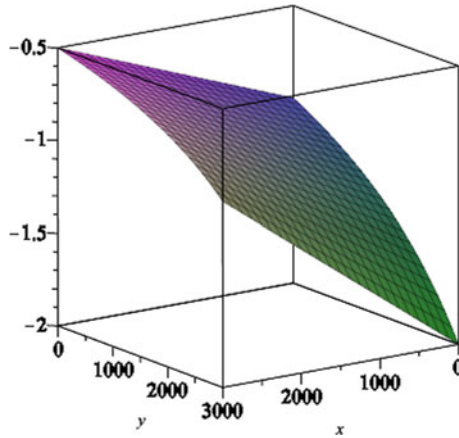


Fig. 7 The tangent line slope for constant ellipse automated market maker

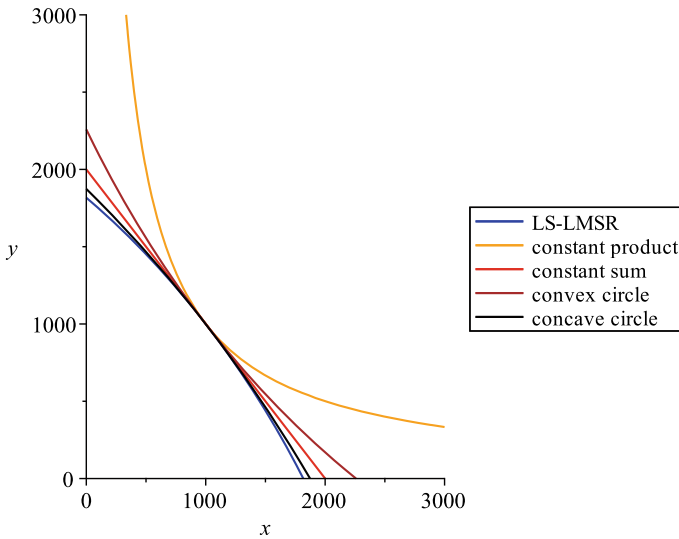


Fig. 8 Cost functions (bottom up): $(x + y) \ln \left(e^{\frac{x}{x+y}} + e^{\frac{y}{x+y}} \right) = 2000 \cdot \ln (2e^{1/2})$, $(x + 6000)^2 + (y + 6000)^2 = 2 \times 7000^2$, $x + y = 2000$, $(x - 6000)^2 + (y - 6000)^2 = 2 \times 5000^2$, and $xy = 1000000$

4.4 Front Running Attacks Based on Slippage

Slippage based front-running attacks can always be launched if the tangent line slope for the cost function curve is not a constant. The more the tangent line slope fluctuates around the current market state, the more profit the front-runner can make. The analysis in preceding sections show that tangent line slopes for LS-LMSR and

constant ellipse cost functions fluctuate smoothly and tangent line slopes for constant product/mean cost functions fluctuate sharply. Thus LS-LMSR and constant ellipse cost function automated markets are more robust against front running attacks. In Uniswap V2, when a trader submits a transaction buying coins of token A with coins of token B (or vice versa), the trader may submit the order at the limit. But the front runner can always try to profit by letting the trader's order be executed at the limit price as shown in the following attacks against Uniswap V2.

Example 1 Most front running attacks leverage off-chain bots and on-chain proxy smart contracts to carry out attacks (see., e.g, [10]). There are some statistics on these front running bots at Dune Analytics (see, e.g., [1]). The following two recent proxy smart contracts take advantage of the large slippage on Uniswap V2.

- [0xd59e5b41482ee6283c22e1a6a20756da512ffa97](#) received a profit of at least 1,172,436 USD during a 14 days period.
- [0x00000005736775feb0c8568e7dee77222a26880](#) received a profit of 60 ETH during one week. The profit was transferred to another address [0x94dD...](#)

We analyze attacking steps by the second bot against Uniswap V2 Pair SPA-ETH: [0x13444ec1c3ead70ff0cd11a15bfdc385b61b0fc2](#). The attacking transactions are included in the block [12355902](#) finalized on May-02-2021 04:43:18 PM.

1. The attacker saw that [0x006fa275887292cdc07169f1187b7474e376bb3b](#) submitted an order to swap 4.544 ETH for SPA.
2. The attacker's smart contract inserts an order to swap 2.6842 ETH for 385,583 SPA before the above observed order in transaction hash [0x4e2636...](#)
3. [0x006fa...](#)'s order is fulfilled at the transaction hash [0x9a17...](#) where the user received 613,967 SPA for his 4.544 ETH.
4. The attacker's smart contract inserts an order to swap 385,583 SPA for 2.8778 ETH after the above observed order in transaction hash [0x34787e...](#)
5. The attacker's smart contract received 0.1936 ETH for free.

5 Price Amplitude

For constant product/mean AMMs, the relative price $\frac{P_1(\mathbf{q})}{P_2(\mathbf{q})}$ of the two tokens ranges from 0 (not inclusive) to ∞ . At the time when a tiny portion of one token coin is equivalent to all coins of the other token in the market maker, no trade is essentially feasible. Thus the claimed advantage that no one can take out all shares of one token from the constant product/mean market seems to have limited value. For a given LS-LMSR (or constant ellipse) automated market with an initial state \mathbf{q}_0 , the relative price $P_1(\mathbf{q})/P_2(\mathbf{q})$ can take values only from a fixed interval. If the market changes and this relative price interval no longer reflects the market price of the two tokens, one may need to add tokens to the market to adjust this price interval. On the other hand, it may be more efficient to just cancel this automated market maker and create a new AMM when this situation happens.

In the following example, we show how to add liquidity to an existing LS-LMSR AMM to adjust the relative price range. Assume that the market price for a coin of token A is 100 times the price for a coin of token B when the AMM is incorporated. The patron uses 10 coins of token A and 1000 coins of token B to create an AMM with the initial state $\mathbf{q}_0 = (1000, 1000)$. The total market cost is $C(\mathbf{q}_0) = 2386.294362$. Assume that after some time, the AMM moves to state $\mathbf{q}_1 = (100, 1750.618429)$. At \mathbf{q}_1 , we have $P_1(\mathbf{q}_1)/P_2(\mathbf{q}_1) = 0.6809820540$ which is close to the lowest possible value 0.6481149479. In order to adjust the AMM so that it still works when the value P_1/P_2 in the real world goes below 0.6481149479, the patron can add some coins of token A to \mathbf{q}_1 so that the resulting market state is $\mathbf{q}_2 = (1750.618429, 1750.618429)$. To guarantee that one coin of token B is equivalent to $\frac{P_2(\mathbf{q}_1)}{100 \cdot P_1(\mathbf{q}_1)} = 0.01468467479$ coins of token A in \mathbf{q}_2 , we need to have the following mapping from outstanding shares in \mathbf{q}_2 to actual token coins (note that this mapping is different from that for \mathbf{q}_0):

- Each outstanding share of token A corresponds to 0.01468467479 coin of token A .
- Each outstanding share of token B corresponds to one coin of token B .

Thus there are $1750.618429 \times 0.01468467479 = 25.70726231$ coins of token A in \mathbf{q}_2 . Since there is only one coin of token A in \mathbf{q}_1 , the patron needs to deposit 24.70726231 coins of token A to \mathbf{q}_1 to move the AMM to state \mathbf{q}_2 . If the market owner chooses not to deposit these tokens to the market, the market maker will still run, but there is a chance that the outstanding shares of token A goes to zero at certain time.

In the above scenario, one may ask whether it is possible for the market maker to automatically adjust the market state to $\mathbf{q}_3 = (1750.618429, 1750.618429)$ by re-assigning the mapping from shares to coins? If \mathbf{q}_2 automatically adjusts itself to \mathbf{q}_3 without external liquidity input, then a trader may use one share of token A to get one share of token B in \mathbf{q}_3 . Since we only have one equivalent coin of token A but 1750.618429 outstanding shares in \mathbf{q}_3 , each outstanding share of token A in \mathbf{q}_3 is equivalent to 0.0005712267068 coins of token A . That is, the trader used 0.0005712267068 coins of token A to get one coin of token B (note that each outstanding share of token B corresponds to one coin of token B in \mathbf{q}_3). By our analysis in the preceding paragraphs, at \mathbf{q}_3 , one coin of token B has the same market value of 0.01468467479 coins of token A . In other words, the trader used 0.0005712267068 coins of token A to get equivalent 0.01468467479 coins of token A . Thus it is impossible for the automated market to adjust its relative price range without an external liquidity input.

6 Implementation and Performance

We have implemented the constant ellipse based AMMs using Solidity smart contracts and have deployed them over the Ethereum blockchain. The smart contract source codes and Web User Interface are available at GitHub. As an example, we use

the ellipse $(x - c)^2 + (y - c)^2 = r^2$ to show how to establish a token pair swapping market in this section. Specifically, we use $c = 10^9$ and $r \cdot 10^{14} = 16000 \cdot 10^{14}$ (that is, $r = 16000$) for illustration purpose in this section.

Each token pair market maintains constants λ_0 and λ_1 which are determined at the birth of the market. Furthermore, each token market also maintains a non-negative multiplicative scaling variable μ which is the minimal value so that the equation $(\mu\lambda_0x_0 - 10^9)^2 + (\mu\lambda_1y_0 - 10^9)^2 \leq 16000 \cdot 10^{14}$ holds where $\mu\lambda_0x_0 < 10^9$ and $\mu\lambda_1y_0 < 10^9$. This ensures that we use the lower-left section of the ellipse for the automated market.

6.1 Gas Cost and Comparison

We compare the gas cost against Uniswap V2 and Uniswap V3. During the implementation, we find out that some of the optimization techniques that we used in Coinswap may be used to reduce the gas cost in Uniswap V2. Thus we compare the gas cost for Uniswap V2 (column Uni V2) our optimized version of Uniswap V2 (column Uni V2O), Uniswap V3 (column Uni V3), and our CoinSwap in Table 2. In a summary, our constant ellipse AMM (CoinSwap) has a gas saving from 0.61% to 46.99% over Uniswap V2 and has a gas saving from 23.19% to 184.29% over Uniswap V3. It should be noted that Uniswap V3 tried to reduce the slippage in certain categories though still do not have the full slippage control as CoinSwap has. The testing script that we have used will be available on the Github. Some field for Uniswap V3 in Table 2 is empty since we did not find an easy way to test that in the Uniswap V3 provided testing scripts.

Table 2 Gas cost Uniswap V2, V3, and CoinSwap with liquidity size (40000000,10000000)

Function	UNI V2	UNI V2O	UNI V3	CoinSwap	Saving over UNI V2	Saving over UNI V3
mint()	141106	132410	308610	109722	28.60%	184.29%
swap()	89894	88224	114225	89348	0.61%	27.84%
swap()[1st]	101910	100051		96294	5.83%	
add Ω	216512	207368		185442	16.76%	
remove Ω	98597	97319	82694	67127	46.88%	23.19%
add ETH	223074	213930		192027	16.14%	
full removal	123339	122061		98805	24.83%	
partial removal	180355	137061		144283	25.00%	

7 Conclusion

The analysis in the paper shows that constant ellipse cost functions have certain advantages for building AMMs in Decentralized Finance (DeFi) applications. One may argue that constant ellipse cost function based markets have less flexibility after the market is launched since the price amplitude is fixed. We have mentioned that, though the token price could range from 0 to ∞ in the constant product cost model, when the price for one token is close to infinity, any meaningful trade in the market is infeasible. Thus the old market needs to be stopped and a new market should be incorporated. Indeed, it is an advantage for an AMM to have a fixed price amplitude when it is used as a price oracle for other DeFi applications. For the constant product cost market, if the patron incorporates the AMM by depositing a small amount of liquidity, an attacker with a small budget can manipulate the token price significantly in the AMM and take profit from other DeFi applications that use this AMM as a price oracle. For constant ellipse based AMMs, the patron can use a small amount of liquidity to set up the automated market and the attacker can only manipulate the token price within the fixed price amplitude.

References

1. Dune Analytics. (2021). Collected-bot per month. <https://duneanalytics.com/queries/14859>.
2. Fama, E. F. (1970). Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2), 383–417.
3. Good, I. J. (1952). Rational decisions. *J. Royal Statistical Society B*, 14(1), 107–114.
4. Hanson, R. (2003). Combinatorial information market design. *Information Systems Frontiers*, 5(1), 107–119.
5. Hanson, R. (2007). Logarithmic markets coring rules for modular combinatorial information aggregation. *The Journal of Prediction Markets*, 1(1), 3–15.
6. Hertzog, E., Benartzi, G., & Benartzi, G. (2017). Bancor protocol: continuous liquidity for cryptographic tokens through their smart contracts. https://storage.googleapis.com/website-bancor/2018/04/01ba8253-bancor_protocol_whitepaper_en.pdf.
7. Leshner, R., & Hayes, G. (2019). Compound: The money market protocol. <https://compound.finance/documents/Compound.Whitepaper.pdf>.
8. Martinelli, F., & Mushegian, N. (2019). A non-custodial portfolio manager, liquidity provider, and price sensor. <https://balancer.finance/whitepaper/>.
9. Othman, A., Pennock, D. M., Reeves, D. M., & Sandholm, T. (2013). A practical liquidity-sensitive automated market maker. *ACM TEAC*, 1(3), 1–25.
10. Robinson, D., & Konstantopoulos, G. (2020). Ethereum is a dark forest. <https://medium.com/@danrobinson/>.
11. Uniswap. (2020). Uniswap v2 core. <https://uniswap.org/whitepaper.pdf>.
12. Uniswap. (2021). V3. <https://uniswap.org/whitepaper-v3.pdf>.
13. Yongge, W. (2020). Automated market makers for decentralized finance (defi). *arXiv preprint arXiv:2009.01676*.
14. Zhou, L., Qin, K., Torres, C. F., Le, D. V., & Gervais, A. High-frequency trading on decentralized on-chain exchanges.

Wombat—An Efficient Stableswap Algorithm



Jen Hounɡ Lie , Tony W. H. Wong , and Alex Yin-ting Lee 

Abstract Curve Finance invented the first stableswap-focused algorithm. However, its algorithm involves (1) solving complex polynomials and (2) requiring assets in the pool to have the same size of liquidity. This paper introduces a new stableswap algorithm—Wombat, to address these issues. Wombat uses a closed-form solution, so it is more gas efficient and adds the concept of asset-liability management to enable single-side liquidity provision, which increases capital efficiency. Furthermore, we derive efficient algorithms from calculating withdrawal or deposit fees as an arbitrage block. Wombat is named after the short-legged, muscular quadrupedal marsupials native to Australia. As Wombats are adaptable and habitat-tolerant animals, the invariant created is also adaptable and tolerant to liquidity changes.

Keywords Decentralized finance · Automated market maker · Stableswap · Asset liability management · Smart contract

1 Introduction

Automated Market Maker (AMM) is essential for Decentralized Finance (DeFi) traders to swap tokens on-chain. As one of the essential foundations of DeFi, AMM is integral to the success of any blockchain, whether it be the past, present, or future. An efficient AMM will have powerful effects on the ecosystem providing decentralized apps with an efficient platform to build upon and thus, fueling the next evolution of finance. Efforts from Uniswap, Curve, and Balancer have contributed to developing a quick, efficient, and effective AMM design. Most of the existing AMM designs use an invariant curve to create an efficient swap with reasonable slippage directed

J. H. Lie · A. Y. Lee
The University of Hong Kong, Hong Kong SAR, China

T. W. H. Wong
Kutztown University of Pennsylvania, Kutztown, PA, US

T. W. H. Wong (✉) · A. Y. Lee
Wombat Exchange, Hong Kong, China
e-mail: wong@kutztown.edu

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2023
P. Pardalos et al. (eds.), *Mathematical Research for Blockchain Economy*, Lecture Notes in Operations Research, https://doi.org/10.1007/978-3-031-18679-0_13

at assets with a range of volatility characteristics. Uniswap, one of the pioneers of AMM design, designed an AMM that focused on more volatile assets such as ETH and BTC. The invariant curve was designed to be more parabolic to ensure that price change follows demand shock [2]. Noting the characteristics of pegged assets, Curve proposed an invariant curve that was more linear in the interior part of the curve but more parabolic when x or y tended to infinity [3]. Such a design allows a more efficient swap between pegged assets such as stablecoins.

As DeFi pushes toward higher efficiency, one of the main issues with the current designs is their ability to scale while simultaneously remaining capital efficient. These scaling issues are bounded by a protocol's inherent design and result in complicated token-pair compositions. It must be noted that computational inefficiencies affect the end-users in terms of fees forfeited as the cost is proportional to complexity. On top of the cost, users are forced to deal with complex user interfaces due to the inflexible algorithm, something that Wombat can both simplify and make cost-effective. The development of Wombat aims at resolving such deficiencies.

The invariant curve is a relationship between two or more token amounts. Mathematically speaking, it is a level set of a function that maps token amounts to a real number [1]. To simplify the discussion, we focus on the two-token case first. For example, consider $x + y = k$ for some constant k . That means, whenever a swap of Δx to Δy happens on this invariant curve, we know that $x_0 + \Delta x + y_0 + \Delta y = k$, which implies $\Delta x = -\Delta y$. On the xy -plane, the invariant curve is a linear line. The invariant function $x + y = k$ is called the Constant Sum Market Maker (CSMM). Under CSMM, x could be swapped for the same amount of y unless y runs out of liquidity. Another example of an invariant curve would be $xy = k$ for some constant k , also known as Constant Product Market Maker (CPMM) used by Balancer and Uniswap [2]. The invariant curve on xy -plane is more hyperbolic compared to CSMM.

One main problem of many existing AMMs is that they require injecting tokens in pairs [5]. The ratio of these two tokens would, in turn, represent the price ratio, and the functional behavior of the CPMM would control the price movement. The screenshot from Uniswap (see Fig. 2 in Appendix A) illustrates that a user needs to add ETH and USDC to the pool simultaneously.

In the case of Curve Finance, the equilibrium is achieved when the assets inside the pool have the same liquidity amount. Otherwise, the invariant curve will become more parabolic and inherently imply different trading prices for assets within the pool. The screenshot from Curve (see Fig. 3 in Appendix A) shows that USDT is trading at a discount compared to USDC and DAI due to the pool imbalance.

In the world of cryptocurrencies, there is a bias towards specific stablecoins. At the time of writing, the market capitalization for USDT, USDC, and DAI are 80B, 53B, and 6.6B, respectively. With Curve's setup, the pool size will be constrained by the asset with the lowest supply, i.e., DAI, since it is uneconomic to deposit USDT and USDC further when the DAI supply is saturated. For this reason, it will be beneficial to remove the liquidity constraint to increase the system's scalability. Following Wombat's solution, we can have a more efficient algorithm that is more capital and gas efficient while removing scalability issues.

Lastly, Curve’s stableswap algorithm is computationally inefficient when more assets are in the pool. The required gas is summarized below in Sect. 2.3.

2 Wombat’s Design

In light of the problems above, Wombat’s design aims to:

1. Make the algorithm more gas efficient;
2. Allow single-sided liquidity provision;
3. Get rid of the same-liquidity constraint for assets in the same pool.

2.1 Wombat’s Invariant Curve

Stableswap concerns the swap of pegged tokens and assets. Denote the set of tokens with the same peg as

$$\mathcal{T} := \{\text{token } k : V(\text{token } k) = C\},$$

where C is some fixed real number and $V(\cdot)$ is a value function that maps a token to its value metric, usually in USD.

The main goal of a stableswap invariant is to create a function to mimic CSMM when x_k are close to each other. Revisit the Curve’s stableswap invariant:

$$An^n \sum_{k \in \mathcal{T}} x_k + D = ADn^n + \frac{D^{n+1}}{n^n \prod_{k \in \mathcal{T}} x_k},$$

where $A > 0$ denotes the *amplification factor* and D is the level preset by the state of token pools, which is a constant with respect to swap.

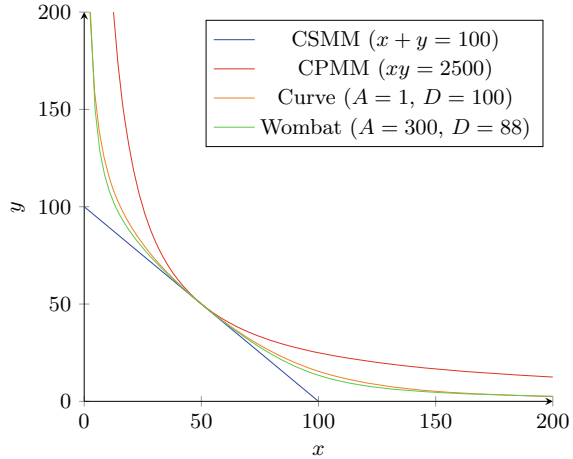
Here, Wombat introduces a new algorithm that can achieve a similar result as Curve’s stableswap invariant, yet much easier to solve:

$$\sum_{k \in \mathcal{T}} \left(x_k - \frac{A}{x_k} \right) = D. \tag{1}$$

The intuition is that when x_k are close to each other, the reciprocal terms will be minimized, making the function closely mimic the linear term, i.e., the CSMM. However, if x_k are apart from each other, the reciprocal terms will grow faster and create a steeper slope for the curve.

For Curve Finance, solving the invariant constant D involves solving a high-degree polynomial. As for Wombat, solving D is plain simple by calculating the summation.

Fig. 1 CFMM comparisons



To perform a swap, Curve would involve solving a function with product terms, while Wombat can be simply solved with a quadratic equation.

Figure 1 plots the invariant curves of CSMM, CPMM, and those of Curve and Wombat. As illustrated in the graph above, the invariant curve of Wombat is similar to that of Curve, meaning that Wombat achieves similar results as Curve with a more efficient algorithm.

2.2 Enhanced Wombat’s Invariant Curve

While Eq. (1) has presented a better alternative to Curve’s stableswap invariant, Wombat further adopts the asset-liability management concept proposed by Platypus Finance [5] to get rid of liquidity constraints. The coverage ratio is used as the input for the invariant curve, which allows single-sided liquidity injection and arbitrary exchange across tokens within the preset pool.

To achieve so, we first forgo the ability to represent token price ratios since they are exogenously set to be 1. The moving parameters of the CFMM need not be the token’s liquidity in the pool; instead, we change them into the coverage ratios r of the tokens, that is,

$$r_k = \frac{A_k}{L_k},$$

where r_k is the *coverage ratio* of token k pool, A_k is the *asset* available for withdrawal and swap in token k pool, and L_k is the *liability* in token k pool. We can also understand the liability L_k as the amount of token k injected in the pool as a deposit, and they are subject to withdrawal at any given time.

Table 1 Comparison of algorithm gas costs

Pool size	Wombat	Curve finance
2	125,713	115,763
3		120,036
4		125,263
5		131,280
6		136,881

Now, we define our modified CFMM as follows: the invariant curve is given by

$$\sum_{k \in \mathcal{T}} L_k \left(r_k - \frac{A}{r_k} \right) = D.$$

Since the equation is now revolving around coverage ratios of tokens instead of liquidities, the system can get rid of the liquidity constraint to enable features such as single-sided deposit and heterogeneous liquidity for equilibrium.

2.3 Gas Comparison

Wombat's simplicity enables it to be more gas efficient. A notable feature for Wombat is that its gas is independent of the pool size, while for Curve, the gas increases as more assets are added into the same pool (see Table 1).

Since Wombat adopts the Asset Liability Management (ALM), additional states need to be written on the blockchain, which is relatively expensive in gas. Nevertheless, Wombat starts to outperform Curve when pool size is larger than 4. It should also be noted that since Curve's algorithm involves solving a high-degree polynomial with brute force, the upper-bound for their gas limit can get very expensive, whereas the upper gas limit for Wombat is known since the solution has a closed form.

2.4 Swap Mechanism

For simplicity, we will focus on the two-token case, which is going to be the primary action performed by the traders. However, it is also worth-mentioning that our design allows a series of swaps simultaneously (i.e., from n tokens to m tokens) as long as it fits the CFMM.

Consider two tokens, i and j . We look into the case when a trader swaps from token i to token j . Let the initial coverage ratios of the two pools be $r_{i,0}$ and $r_{j,0}$, respectively. With the swap, the coverage ratios of the two pools change from $(r_{i,0}, r_{j,0})$ to (r_i, r_j) , where

$$r_i = r_{i,0} + \frac{\Delta_i}{L_i} \quad \text{and} \quad r_j = r_{j,0} + \frac{\Delta_j}{L_j}. \quad (2)$$

Here, $\Delta_i > 0$ is specified by the trader, while an unknown Δ_j is to be determined by the CFMM. Since the level set specified by the CFMM remains unchanged before and after the swap, we have

$$\sum_{k \in \mathcal{T} \setminus \{i,j\}} L_k \left(r_k - \frac{A}{r_k} \right) + L_i \left(r_{i,0} - \frac{A}{r_{i,0}} \right) + L_j \left(r_{j,0} - \frac{A}{r_{j,0}} \right) = D = \sum_{k \in \mathcal{T}} L_k \left(r_k - \frac{A}{r_k} \right).$$

If we define the constant $D_{i,j} = L_i \left(r_{i,0} - \frac{A}{r_{i,0}} \right) + L_j \left(r_{j,0} - \frac{A}{r_{j,0}} \right)$, then

$$L_i \left(r_i - \frac{A}{r_i} \right) + L_j \left(r_j - \frac{A}{r_j} \right) = D_{i,j} \tag{3}$$

After a series of algebraic operations, we obtain the function

$$r_j = \frac{-b + \sqrt{b^2 + 4A}}{2},$$

where

$$b = \frac{L_i}{L_j} \left(r_i - \frac{A}{r_i} \right) - \frac{D_{i,j}}{L_j}.$$

Note that we reject the solution that $r_j < 0$. Knowing $r_{j,0}$ and r_j yields us knowledge on Δ_j .

Next, we focus on the properties of the invariant curves. Using implicit differentiation on Eq. (3), we can obtain the derivative

$$\frac{dr_j}{dr_i} = - \frac{L_i \left(1 + \frac{A}{r_i^2} \right)}{L_j \left(1 + \frac{A}{r_j^2} \right)} < 0.$$

The negative derivative tells us that given the same invariant curve, any swap action that increases r_i would reduce r_j . In other words, the more token i is injected into the system, the more token j the trader receives. Without explicitly calculating the second-order derivative, we can study the behavior of the first-order derivative by noticing that an increase in r_i leads to a decrease in $L_i \left(1 + \frac{A}{r_i^2} \right)$, and a decrease in r_j leads to an increase in $L_j \left(1 + \frac{A}{r_j^2} \right)$. Hence, $\frac{dr_j}{dr_i}$ would become less negative when r_i increases. This implies that $\frac{d^2 r_j}{dr_i^2} > 0$. A convex invariant curve indicates that continuous injection of token i would yield the trader fewer and fewer token j in return per unit of token i injected.

2.5 Slippage Analysis

In the remainder of this section, we take a closer look at the slippage behavior of the invariant curve. Recalling that r_i and r_j relate to Δ_i and Δ_j as described in Eq. (2), we have

$$\frac{d\Delta_j}{d\Delta_i} = \frac{d\Delta_j}{dr_j} \cdot \frac{dr_j}{dr_i} \cdot \frac{dr_i}{d\Delta_i} = L_j \cdot \left(-\frac{L_i \left(1 + \frac{A}{r_i^2}\right)}{L_j \left(1 + \frac{A}{r_j^2}\right)} \right) \cdot \frac{1}{L_i} = -\frac{1 + \frac{A}{r_i^2}}{1 + \frac{A}{r_j^2}}. \quad (4)$$

If $r_i = r_j$, then

$$\left. \frac{d\Delta_j}{d\Delta_i} \right|_{r_i=r_j} = -1,$$

which indicates that every token i injected would yield one token j at this particular state. If $r_i < r_j$, then the derivative in Eq. (4) is less than -1 , so every token i injected yields more than one token j . On the other hand, if $r_i > r_j$, then every token i injected yields less than one token j . Notice that swapping token i to token j increases r_i and lowers r_j . Hence, when $r_i < r_j$, the arbitrage opportunity encourages traders to swap token i to token j , thus reducing the gap between the two coverage ratios. Conversely, if $r_i > r_j$, swapping token i to token j widens the gap between the two coverage ratios and is discouraged with a loss for the traders.

Theorem 1 *The equilibrium state of the AMM is that for all $i, j \in \mathcal{T}$, $r_i = r_j$.*

Proof Assuming the contrary, there exist $i, j \in \mathcal{T}$ such that $r_i < r_j$. Then one can find an arbitrage by swapping token i to token j until $r_i = r_j$.

3 Desirable Properties of AMMs

There are three desirable properties of AMMs, including path independence, liquidity sensitivity, and no-arbitrage [4]. Wombat AMM is capable to achieve path independence and is liquidity sensitive by design.

3.1 Path Independence

Path independence is crucial for AMMs because there should always exist a path to return from any state to the equilibrium in the system. The importance lies in the reality that if the system cannot regain equilibrium, then the system is out of balance and there may be arbitrage opportunities that will negatively impact the system's stability. Path independence ensures that a trader cannot place a series of trades and

profits without assuming a potential risk. Furthermore, it helps provide a minimum representation of the current state in which we are only required to know the quantity vector. Finally, in a path independent system, traders do not need to discover a strategy on how they trade.

Any swap action is performed along the same invariant curve, and the liabilities of all tokens remain unchanged throughout the process. Hence, after the coverage ratios change from $(r_{i,0}, r_{j,0})$ to (r_i, r_j) as described in Eq. (2), the coverage ratios can be returned to the status quo by swapping $-\Delta_j$ token j to token i . Therefore, the Wombat AMM is path independent.

3.2 Liquidity Sensitivity

Liquidity sensitivity is imperative because it makes the protocol desirable in its function, where a trade moves prices less in a liquid market than in an illiquid one. This is important in any trading market, where small trades should not drastically affect the market where sufficient liquidity is present. With this design, our AMM is able to mimic and emulate highly liquid traditional markets where prices can be held stable and more resistant to small trades, enabling a fairer and more transparent platform.

Recall that r_j can be written explicitly as a function of r_i , namely

$$r_j = \frac{-b + \sqrt{b^2 + 4A}}{2},$$

where

$$b = \frac{L_i}{L_j} \left(r_i - \frac{A}{r_i} \right) - \frac{D_{i,j}}{L_j}$$

and

$$D_{i,j} = L_i r_i + L_j r_j - A \left(\frac{L_i}{r_i} + \frac{L_j}{r_j} \right) = L_i r_{i,0} + L_j r_{j,0} - A \left(\frac{L_i}{r_{i,0}} + \frac{L_j}{r_{j,0}} \right).$$

Note that A , r_i , $r_{i,0}$, L_i , and $r_{j,0}$ are independent of L_j while r_j and $D_{i,j}$ are dependent of L_j . Also, recall from Eq. (2) that $\Delta_j = (r_j - r_{j,0})L_j < 0$ in the swap of token i to token j . Now we consider the partial derivative of Δ_j with respect to L_j .

$$\begin{aligned}
 \frac{\partial \Delta_j}{\partial L_j} &= \frac{\partial r_j}{\partial L_j} L_j + r_j - r_{j,0} \\
 &< \frac{1}{2} \frac{\partial b}{\partial L_j} \left(\frac{b}{\sqrt{b^2 + 4A}} - 1 \right) L_j \\
 &= \frac{1}{2} \left(-\frac{1}{L_j^2} \left(L_i \left(r_i - \frac{A}{r_i} \right) - D_{i,j} \right) + \frac{1}{L_j} \left(r_{j,0} - \frac{A}{r_{j,0}} \right) \right) \left(\frac{b}{\sqrt{b^2 + 4A}} - 1 \right) L_j \\
 &= \frac{1}{2} \left((r_{j,0} - r_j) - A \left(\frac{1}{r_{j,0}} - \frac{1}{r_j} \right) \right) \left(\frac{b}{\sqrt{b^2 + 4A}} - 1 \right) < 0.
 \end{aligned}$$

The last inequality holds since $r_j < r_{j,0}$ and $\frac{b}{\sqrt{b^2 + 4A}} < 1$. Hence, if we keep $r_i, r_{i,0}, L_i,$ and $r_{j,0}$ unchanged, then Δ_j becomes more negative when L_j grows, meaning that the yield of token j increases.

4 Arbitrage Block

Wombat’s design suffers from arbitrage issues when a withdrawal or deposit is made. The arbitrage issue is partly due to the asset-liability management design, as there are opportunities for the coverage ratios to be manipulated by rogue traders. A simple numerical example is illustrated below (assume $A = 0.01$).

There are two tokens, X and Y, whose assets and liabilities start at 100. The attacker first swaps 50 units of token X for 49.36 units of token Y, and the coverage ratio of token Y drops from 1 to 0.51.

The attacker is also a major liquidity provider for token Y. He withdraws 50 unit of token Y, and the coverage ratio of token Y drops from 0.51 to 0.013.

Finally, the attacker swaps the 49.36 units of token Y he obtained from Step 1 back to token X. Note that the attacker gets a better exchange rate in Step 3 than Step 1 because the coverage ratio is being manipulated. The attacker earns $(49.36 - 50) - (49.36 - 86.92) = \36.92 from the whole operation.

Therefore, the goal for this section is to provide an algorithm to block the arbitrage and solve it mathematically (Tables 2, 3 and 4).

Table 2 Swap X for Y

Asset	Liability
Token X: 100	Token X: 100
Token Y: 100	Token Y: 100
Asset	Liability
Token X: 150	Token X: 100
Token Y: 50.64	Token Y: 100

Table 3 Withdraw Y

Asset	Liability
Token X: 150	Token X: 100
Token Y: 50.64	Token Y: 100
Asset	Liability
Token X: 150	Token X: 100
Token Y: 0.64	Token Y: 50

Table 4 Swap Y for X

Asset	Liability
Token X: 150	Token X: 100
Token Y: 0.64	Token Y: 50
Asset	Liability
Token X: 63.08	Token X: 100

4.1 Changes in the Asset and Liability

Recall once again that our modified CFMM is

$$\sum_{k \in \mathcal{T}} L_k \left(r_k - \frac{A}{r_k} \right) = D.$$

If the system returns to the equilibrium state as described by Theorem 1, then the global equilibrium coverage ratio r^* satisfies

$$\begin{aligned} \sum_{k \in \mathcal{T}} L_k \left(r^* - \frac{A}{r^*} \right) &= D & (5) \\ \left(\sum_{k \in \mathcal{T}} L_k \right) (r^*)^2 - Dr^* - A \left(\sum_{k \in \mathcal{T}} L_k \right) &= 0 \\ (r^*)^2 - \frac{D}{\sum_{k \in \mathcal{T}} L_k} r^* - A &= 0. \end{aligned}$$

Solving the quadratic equation, we obtain $r^* = \frac{-b + \sqrt{b^2 + 4A}}{2}$, where $b = -\frac{D}{\sum_{k \in \mathcal{T}} L_k}$.

In a withdrawal or deposit of token i , let δ_i^A and δ_i^L denote the change in the asset and the change in the liability, respectively. Here, $\delta_i^L \geq -L_i$ is specified by the trader,

where $\delta_i^L < 0$ denotes a withdrawal and $\delta_i^L > 0$ denotes a deposit. The quantity δ_i^A is to be solved as a function of δ_i^L to block the arbitrage. The new global equilibrium coverage ratio $r^{*'}$ after the withdrawal or deposit is given by

$$r^{*'} = \frac{\delta_i^L + (\sum_{k \in \mathcal{T}} L_k) r^*}{\delta_i^L + \sum_{k \in \mathcal{T}} L_k}. \tag{6}$$

In this equation, $(\sum_{k \in \mathcal{T}} L_k) r^*$ denotes the total asset if the system returns to the equilibrium state. By adding δ_i^L , the numerator expresses the total asset after withdrawal or deposit if the system first returns to the equilibrium state, while the denominator denotes that total liability after withdrawal or deposit. From $r^{*'}$, we can solve for the new constant D' such that

$$\left(\delta_i^L + \sum_{k \in \mathcal{T}} L_k \right) \left(r^{*' } - \frac{A}{r^{*' }} \right) = D'. \tag{7}$$

With the new constant D' , we can backward deduce the coverage ratio r'_i needed to maintain the equilibrium of the system:

$$(L_i + \delta_i^L) \left(r'_i - \frac{A}{r'_i} \right) + \sum_{k \in \mathcal{T} \setminus \{i\}} L_k \left(r_k - \frac{A}{r_k} \right) = D' \tag{8}$$

$$\begin{aligned} (L_i + \delta_i^L)(r'_i)^2 + \left(\sum_{k \in \mathcal{T} \setminus \{i\}} L_k \left(r_k - \frac{A}{r_k} \right) - D' \right) r'_i - A(L_i + \delta_i^L) &= 0 \\ (r'_i)^2 + \frac{1}{L_i + \delta_i^L} \left(\sum_{k \in \mathcal{T} \setminus \{i\}} L_k \left(r_k - \frac{A}{r_k} \right) - D' \right) r'_i - A &= 0, \end{aligned}$$

so

$$r'_i = \frac{-b' + \sqrt{(b')^2 + 4A}}{2},$$

where

$$\begin{aligned} b' &= \frac{1}{L_i + \delta_i^L} \left(\sum_{k \in \mathcal{T} \setminus \{i\}} L_k \left(r_k - \frac{A}{r_k} \right) - D' \right) \\ &= \frac{1}{L_i + \delta_i^L} \left(\sum_{k \in \mathcal{T}} L_k \left(r_k - \frac{A}{r_k} \right) - L_i \left(r_i - \frac{A}{r_i} \right) - D' \right) \\ &= \frac{1}{L_i + \delta_i^L} \left(D - L_i \left(r_i - \frac{A}{r_i} \right) - D' \right). \end{aligned} \tag{9}$$

Finally, the corresponding change δ_i^A in the asset of token i is given by

$$\begin{aligned} \delta_i^A &= (L_i + \delta_i^L)r_i' - L_i r_i \\ &= \frac{D' + L_i \left(r_i - \frac{A}{r_i}\right) - D + \sqrt{\left(D' + L_i \left(r_i - \frac{A}{r_i}\right) - D\right)^2 + 4A(L_i + \Delta_i)^2}}{2} - L_i r_i. \end{aligned} \quad (10)$$

4.2 Maintain Global Equilibrium with $r^* = 1$

The essence of our modified CFMM is that the system is the most stable when all coverage ratios are 1. Swap, withdrawal, and deposit will change the coverage ratios of each individual pool, but our design guarantees that we can always maintain the global equilibrium coverage ratio as $r^* = 1$, which is explained as follows.

As seen in Sect. 2.4, the liabilities L_k and the constant D stay unchanged in our swap mechanism, so the global equilibrium coverage ratio r^* in

$$\left(\sum_{k \in \mathcal{T}} L_k\right) \left(r^* - \frac{A}{r^*}\right) = D$$

is always preserved. As for withdrawal or deposits, if the initial global equilibrium coverage ratio is $r^* = 1$, then the new global equilibrium coverage ratio r^* , defined in Eq. (6), is also 1.

Maintaining the global equilibrium coverage ratio to be 1 has another significant implication, given by the following theorem. The proof of this theorem is given in Appendix B.

Theorem 2 *Assume that $r^* = 1$. If $\delta_i^L < 0$, then $\delta_i^L \leq \delta_i^A < 0$; if $\delta_i^L > 0$, then $0 < \delta_i^A \leq \delta_i^L$. Furthermore, in both cases, $\delta_i^L = \delta_i^A$ if and only if $r_i = 1$.*

4.3 Withdrawal Fees and Deposit Gains

Now, we are ready to describe our algorithm to block the arbitrage. When a withdrawal is made, $\Delta_i < 0$ is specified by the trader. We define $\delta_i^L = \Delta_i$ as the change in the liability of token i , and δ_i^A is given by

$$\delta_i^A = \frac{L_i \left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1 - A) + \sqrt{\left(L_i \left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1 - A)\right)^2 + 4A(L_i + \delta_i^L)^2}}{2} - L_i r_i. \quad (11)$$

As stated in Theorem 2, $|\delta_i^A| \leq |\delta_i^L| = |\Delta_i|$, so from the trader's perspective, there is always a withdrawal fee unless $r_i = 1$. Furthermore, it is apparent from Eq. (2) that $\delta_i^A \geq -L_i r_i = -A_i$, so the final amount of token i that the trader receives is bounded above by the amount of token i available in the system.

On the other hand, when a deposit is made, $\Delta_i > 0$ is specified by the trader. We define $\delta_i^A = \Delta_i$ as the change in the asset of token i , and we solve for δ_i^L in Eq. (11). Rearranging the terms and letting $A'_i = \delta_i^A + L_i r_i$ be the asset of token i after the deposit, we have

$$\begin{aligned} & 2A'_i - \left(L_i \left(r_i - \frac{A}{r_i} \right) + \delta_i^L (1 - A) \right) \\ &= \sqrt{\left(L_i \left(r_i - \frac{A}{r_i} \right) + \delta_i^L (1 - A) \right)^2 + 4A(L_i + \delta_i^L)^2}. \end{aligned}$$

If we square the equation, cancel the identical terms on both sides, and divide the equation by 4, we get

$$(A'_i)^2 - A'_i \left(L_i \left(r_i - \frac{A}{r_i} \right) + \delta_i^L (1 - A) \right) = A(L_i + \delta_i^L)^2.$$

This is a quadratic equation in δ_i^L , namely

$$A(\delta_i^L)^2 + b\delta_i^L + c = 0,$$

where $b = A'_i(1 - A) + 2AL_i$ and $c = A'_i L_i \left(r_i - \frac{A}{r_i} \right) - (A'_i)^2 + AL_i^2$. The discriminant of the quadratic equation is given by

$$\begin{aligned} b^2 - 4Ac &= (A'_i)^2(1 - A)^2 + 4A'_i(1 - A)AL_i + 4A^2L_i^2 \\ &\quad - 4AA'_iL_i \left(r_i - \frac{A}{r_i} \right) + 4A(A'_i)^2 - 4A^2L_i^2 \\ &= (A'_i)^2(1 + A)^2 + 4AA'_iL_i \left(1 - A - r_i + \frac{A}{r_i} \right), \end{aligned}$$

thus the change in the liability δ_i^L is

$$\delta_i^L = \frac{-(A'_i(1 - A) + 2AL_i) + \sqrt{(A'_i)^2(1 + A)^2 + 4AA'_iL_i \left(1 - A - r_i + \frac{A}{r_i} \right)}}{2A}. \quad (12)$$

The negative branch is rejected since it is less than $-L_i r_i$. As stated in Theorem 2, $\Delta_i = \delta_i^A \leq \delta_i^L$, so from the trader's perspective, there is always a deposit gain unless $r_i = 1$.

5 Swap with Haircut Fees

As a profit for the protocol, we charge a certain percentage for each swap as a *haircut fee*, denoted as h . Part of this fee will be shared with the liquidity provider and will enter the token pool as a deposit; the rest is going to be retained by Wombat. Let ρ denote the *LP dividend ratio*, which is the ratio of the haircut fee that is shared with the liquidity provider. The portion retained by Wombat will not be included as part of the token assets for accounting purpose to maintain $r^* = 1$. In the following, we will describe how swap is performed with haircut fees.

During the first stage, token i is swapped to token j as described in Sect. 2.4. Here, we provide an explicit expression for Δ_j .

$$\begin{aligned} \Delta_j &= L_j r_j - L_j r_{j,0} \\ &= \frac{D_{i,j} - L_i \left(r_i - \frac{A}{r_i} \right) + \sqrt{\left(D_{i,j} - L_i \left(r_i - \frac{A}{r_i} \right) \right)^2 + 4AL_j^2}}{2} - L_j r_{j,0}, \end{aligned}$$

where r_i and $D_{i,j}$ are given by Eqs. (2) and (3), respectively. After the swap, the trader receives $(1-h)|\Delta_j|$ token j , with $h|\Delta_j|$ token j deducted as the haircut fee. Before the haircut fee is redeposited into the system, the amount of assets in token i and token j are respectively

$$L_i r_{i,0} + \Delta_i \quad \text{and} \quad L_j r_{j,0} + \Delta_j,$$

while the liabilities of token i and token j maintain at L_i and L_j , respectively.

Next, $\Delta'_j = h\rho|\Delta_j|$ token j is deposited into the system since it is shared with the liquidity provider. Using the token j version of Eq. (12) and setting $\delta_j^A = \Delta'_j$, we can compute the corresponding δ_j^L . Therefore, the summary of the final amounts is given by the following table (Table 5).

Table 5 Final asset and liability of tokens i and j after a swap with haircut fees

Final asset of token i	$L_i r_{i,0} + \Delta_i$
Final asset of token j	$L_j r_{j,0} + (1-h\rho)\Delta_j$
Final liability of token i	L_i
Final liability of token j	$L_j + \delta_j^L$

6 Exact Swap, Withdraw, and Deposit when $r^* = 1$

The previous sections show that swap, withdraw, and deposit almost always come with fees or gains. Sometimes, it is desirable to deduce the appropriate amount a trader needs to initially swap, withdraw, or deposit to attain the exact target amount.

In a swap, if the trader specifies the exact amount $|d_j|$ of token j that they would like to receive with $d_j < 0$, then $\Delta_j = \frac{d_j}{1-h}$, and r_j is defined as in Eq. (2). After solving for r_i in Eq. (3), we can deduce that

$$\begin{aligned} \Delta_i &= L_i r_i - L_i r_{i,0} \\ &= \frac{D_{i,j} - L_j \left(r_j - \frac{A}{r_j} \right) + \sqrt{\left(D_{i,j} - L_j \left(r_j - \frac{A}{r_j} \right) \right)^2 + 4AL_i^2}}{2} - L_i r_{i,0}, \end{aligned}$$

which is the amount of token i that the trader should swap.

In withdrawal or deposits, as shown in Eqs. (11) and (12), δ_i^A and δ_i^L can be expressed as a function of each other. Hence, if the trader wants to receive exactly $|d_i|$ token i in a withdrawal with $d_i < 0$, then letting $\delta_i^A = d_i$ and solving for δ_i^L using Eq. (12), we obtain $\Delta_i = \delta_i^L$ as the initial parameter for the withdrawal. Similarly, if the trader wants to have exactly d_i token i stored in the system with $d_i > 0$, then letting $\delta_i^L = d_i$ and solving for δ_i^A using Eq. (11), we obtain $\Delta_i = \delta_i^A$ as the initial parameter for the deposit.

7 Conclusions

We have introduced an innovative algorithm that resolves many issues of the current AMM environment. The existing solutions in the market are computationally inefficient, and these protocols lack the ability to scale while keeping an intuitive design both technically and visually. Additional protocols have been built on these platforms to address some of these problems, but they inevitably add an extra layer of complexity for end-users. In the world of blockchain smart-contract development, unnecessary complexity is a burden to the end-users due to high gas fees and disincentivizes users from interacting with the protocol.

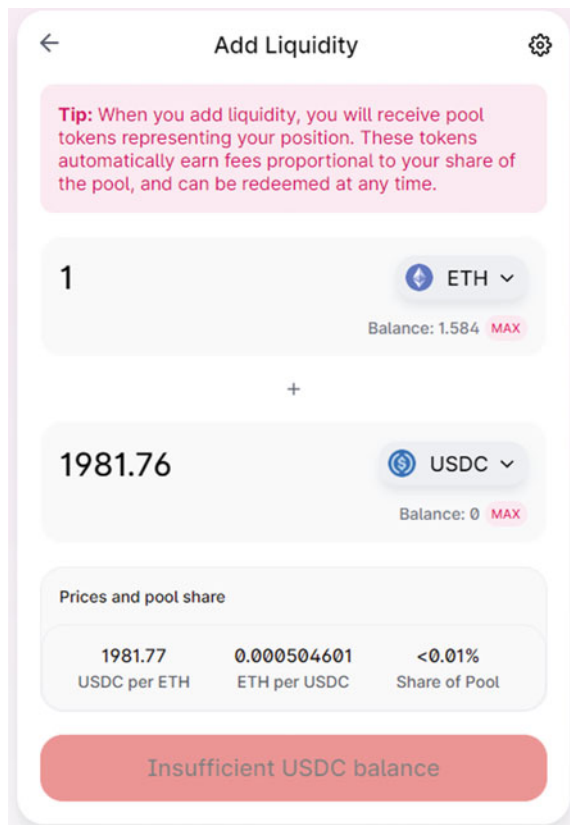
The design of the Wombat algorithm allows for maximum capital efficiency and scalability, which helps promote the growth of decentralized finance. Our work is centered on a CFMM that shows positive homogeneity since the response of relative price is identical at various levels of liquidity. Wombat's algorithm provides a solution

to the status quo, which results in a price-sensitive and path-independent solution while being computationally efficient. We have proved that our arbitrage block can withstand manipulation of the coverage ratio. The arbitrage block can protect the system from malicious attacks, shielding Wombat from attack vectors that would hurt the system.

A Screenshots of Uniswap and Curve

See Figs. 2 and 3.

Fig. 2 Uniswap screenshot



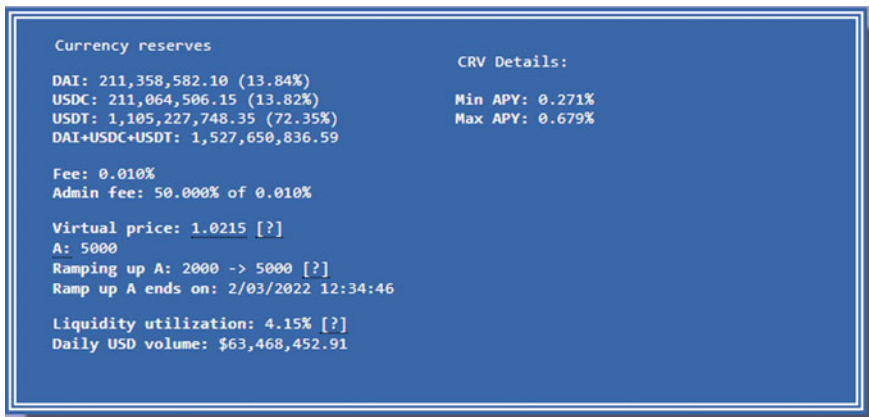


Fig. 3 Curve screenshot

B Proof of Theorem 2

Proof If $r^* = 1$, then $D = (\sum_{k \in \mathcal{T}} L_k) (1 - A)$ by Eq. (5), $r^{*'} = 1$ by Eq. (6), and $D' = (\delta_i^L + \sum_{k \in \mathcal{T}} L_k) (1 - A)$ by Eq. (7). Hence, Eq. (8) becomes

$$\begin{aligned}
 (L_i + \delta_i^L) \left(\frac{A_i + \delta_i^A}{L_i + \delta_i^L} - \frac{A(L_i + \delta_i^L)}{A_i + \delta_i^A} \right) + \sum_{k \in \mathcal{T} \setminus \{i\}} L_k \left(r_k - \frac{A}{r_k} \right) &= \left(\delta_i^L + \sum_{k \in \mathcal{T}} L_k \right) (1 - A) \\
 A_i + \delta_i^A - \frac{A(L_i + \delta_i^L)^2}{A_i + \delta_i^A} + D - L_i \left(r_i - \frac{A}{r_i} \right) &= \delta_i^L (1 - A) + D \\
 \delta_i^A - \frac{A(L_i + \delta_i^L)^2}{A_i + \delta_i^A} + \frac{AL_i^2}{A_i} &= \delta_i^L (1 - A). \tag{13}
 \end{aligned}$$

If $\delta_i^L < 0$ and $\delta_i^A \geq 0$, then the left hand side (LHS) of Eq. (13) is positive while the right hand side (RHS) is negative, a contradiction. Similarly, if $\delta_i^L > 0$ and $\delta_i^A \leq 0$, then the LHS is negative while the RHS is positive, a contradiction again. Hence, δ_i^L and δ_i^A always share the same sign.

To compare δ_i^L and δ_i^A , we first rewrite Eq. (9) as

$$\begin{aligned}
 b' &= \frac{1}{L_i + \delta_i^L} \left(\left(\sum_{k \in \mathcal{T}} L_k \right) (1 - A) - L_i \left(r_i - \frac{A}{r_i} \right) - \left(\delta_i^L + \sum_{k \in \mathcal{T}} L_k \right) (1 - A) \right) \\
 &= -\frac{1}{L_i + \delta_i^L} \left(L_i \left(r_i - \frac{A}{r_i} \right) + \delta_i^L (1 - A) \right).
 \end{aligned}$$

Therefore, Eq. (10) yields

$$\begin{aligned} & \delta_i^A - \delta_i^L \\ &= \frac{-(L_i + \delta_i^L)b' + \sqrt{((L_i + \delta_i^L)b')^2 + 4A(L_i + \delta_i^L)^2}}{2} - L_i r_i - \delta_i^L \\ &= \frac{-L_i \left(r_i + \frac{A}{r_i}\right) - \delta_i^L(1 + A) + \sqrt{\left(L_i \left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1 - A)\right)^2 + 4A(L_i + \delta_i^L)^2}}{2}. \end{aligned}$$

Note that

$$\left(r_i - \frac{A}{r_i}\right)^2 + 4A = r_i^2 - 2A + \frac{A^2}{r_i^2} + 4A = \left(r_i + \frac{A}{r_i}\right)^2$$

and

$$(1 - A)^2 + 4A = 1 - 2A + A^2 + 4A = (1 + A)^2.$$

Furthermore, by the AM-GM inequality, we have $r_i + \frac{1}{r_i} \geq 2$, so

$$\begin{aligned} \left(r_i - \frac{A}{r_i}\right)(1 - A) + 4A &= r_i + \frac{A^2}{r_i} - A \left(r_i + \frac{1}{r_i}\right) + 4A \\ &\leq r_i + \frac{A^2}{r_i} - A \left(r_i + \frac{1}{r_i}\right) + 2A \left(r_i + \frac{1}{r_i}\right) \\ &= \left(r_i + \frac{A}{r_i}\right)(1 + A), \end{aligned}$$

where the equality holds if and only if $r_i = 1$. As a result,

$$\begin{aligned} & \left(L_i \left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1 - A)\right)^2 + 4A(L_i + \delta_i^L)^2 \\ &= L_i^2 \left(r_i - \frac{A}{r_i}\right)^2 + 4AL_i^2 + 2L_i \left(r_i - \frac{A}{r_i}\right) \delta_i^L(1 - A) + 8AL_i \delta_i^L \\ & \quad + (\delta_i^L)^2(1 - A)^2 + 4A(\delta_i^L)^2 \\ &= L_i^2 \left(r_i + \frac{A}{r_i}\right)^2 + 2L_i \delta_i^L \left(\left(r_i - \frac{A}{r_i}\right)(1 - A) + 4A\right) + (\delta_i^L)^2(1 + A)^2. \end{aligned}$$

If $\delta_i^L < 0$, then

$$\begin{aligned}
& \sqrt{\left(L_i \left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1 - A)\right)^2 + 4A(L_i + \delta_i^L)^2} \\
& \geq \sqrt{L_i^2 \left(r_i + \frac{A}{r_i}\right)^2 + 2L_i\delta_i^L \left(r_i + \frac{A}{r_i}\right)(1 + A) + (\delta_i^L)^2(1 + A)^2} \\
& = \left|L_i \left(r_i + \frac{A}{r_i}\right) + \delta_i^L(1 + A)\right|,
\end{aligned}$$

so $\delta_i^A - \delta_i^L \geq 0$, with the equality holds if and only if $r_i = 1$; if $\delta_i^L > 0$, then

$$\begin{aligned}
& \sqrt{\left(L_i \left(r_i - \frac{A}{r_i}\right) + \delta_i^L(1 - A)\right)^2 + 4A(L_i + \delta_i^L)^2} \\
& \leq \sqrt{L_i^2 \left(r_i + \frac{A}{r_i}\right)^2 + 2L_i\delta_i^L \left(r_i + \frac{A}{r_i}\right)(1 + A) + (\delta_i^L)^2(1 + A)^2} \\
& = \left|L_i \left(r_i + \frac{A}{r_i}\right) + \delta_i^L(1 + A)\right|,
\end{aligned}$$

so $\delta_i^A - \delta_i^L \leq 0$, again with the equality holds if and only if $r_i = 1$.

References

1. Angeris, G., & Chitra, T. (2020). Improved price oracles: Constant function market makers. *SSRN Electronic Journal*.
2. Hayden, A. (2019). Uniswap birthday blog–v0.
3. Michael, E. (2019). stableswap–efficient mechanism for stablecoin liquidity.
4. Othman, A., Sandholm, T., Pennock, D. M., & Reeves, D. M. (2013). A practical liquidity-sensitive automated market maker. *ACM Transactions on Economics and Computation (TEAC)*, 1(3), 1–25.
5. Platypus Team. (2021). Platypus AMM technical specification.

Multi-Tier Reputation for Data Cooperatives



Abiola Salau, Ram Dantu, Kirill Morozov, Kritagya Upadhyay,
and Syed Badruddoja

Abstract Data cooperatives allow their members—the data owners—to pool their digital assets together for processing and access management. In this context, reputation is an important measure of trust, which can effectively complement financial assets in the decentralized scenario, also providing incentives for users' honest behavior. We present a decentralized data cooperative system based on the Proof-of-Reputation and Proof-of-Stake blockchains. In order to provide inclusivity for low-reputation (newly joined) users, which is required in our community-based scenario, we use the tier-based committee selection introduced by Kleinrock et al. at Indocrypt 2020. As the underlying Proof-of-Stake system, we use Snow White due to its convenient properties such as flexible committee selection and user participation.

Keywords Blockchain · Data cooperative · Reputation system · Proof-of-Reputation (PoR) · Proof-of-Stake (PoS) · Reputation fairness

1 Introduction

Declining trust between data subjects and data operators, e.g., social media network providers, has been observed in recent years [1]. It was fueled by concerns about user data privacy and, in particular, the improper use of personal information. In turn, this motivated intensive research in the area of *data economy* [2]. Notably, Pentland

A. Salau (✉) · R. Dantu · K. Morozov · K. Upadhyay · S. Badruddoja
Department of Computer Science and Engineering, University of North Texas, Denton, TX
76207, USA
e-mail: abiolasalau@my.unt.edu

R. Dantu
e-mail: ram.dantu@unt.edu

K. Morozov
e-mail: kirill.morozov@unt.edu

K. Upadhyay
e-mail: kritagyaupadhyay@my.unt.edu

S. Badruddoja
e-mail: syedbadruddoja@my.unt.edu

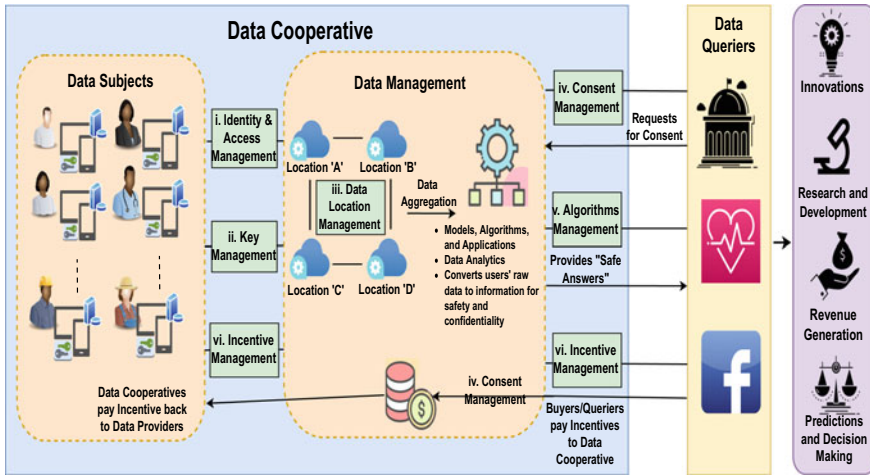


Fig. 1 An overview of a data cooperative [8]

[3] discussed a vision of the new economy, which will be driven by data. This work not only indicated *data* as the backbone of the new economy but also identified flaws in the centralized methods of data management, which lack user-centricity. Further, Pentland [3] proposed the renegotiation of data rights as a mechanism for the data subjects to be in control of their data. Also, he emphasized privacy-preserving machine learning algorithms as well as the distribution and decentralization of data management as key factors to the success of such an economy.

The notion of a *data cooperative*, which results from this concept of decentralized data management, demands user-centric data ownership and shared data as the basis of a new knowledge economy. A data cooperative, as depicted in Fig. 1, can be described as a legal fiduciary where people come together to pool data for the benefit of its members [4, 5]. These data then become their *shared resources*. The protection of personal data pooled together is one of the fundamental challenges in this setting, as identified at the 2019 Data Coops Workshop [6]. Zyskind et al. [7] proposed a decentralized privacy system that adopts the use of blockchain for the protection of personal data. Their work showed that blockchain technology can contribute to data protection in this context.

Furthermore, blockchain technology is important as a means to encourage cooperation and build trust among participants [9]. A natural measure for such trust may be the participants' reputations. The latter may be gauged via a reputation system, which is maintained on top of a blockchain [10]. The associated reputation score of a participant may be adjusted depending on their actions in the system. Reputation can also serve as an asset and/or basis for rewards (such as being selected as a block proposer or endorser), thereby encouraging honest behavior. Note that the trustworthiness of the pooled data may be a concern. This can, however, be well managed by

the blockchain since it is tamper-resistant and any infringement that may affect data integrity would be detected.

For instance, in a neighborhood watch system such as [11], any member who shares misinformation can be traced and detected in the future since the records on the blockchain are immutable. Other applicable security mechanisms include, e.g., Decentralized Identity (DID) [12]¹ and cryptographic mechanisms such as, e.g., data encryption and zero-knowledge proofs, which can provide privacy for data cooperative users' personal data. Moreover, blockchain technology may help to improve security for participants of multiple online platforms or services through the use of decentralized federated identities [13].

Nevertheless, in order to successfully run a data cooperative using blockchain technology, the design of the former has to be properly adapted. In particular, aside from the data owners who share their digital assets, there should be other types of participants who will be responsible for maintaining the blockchain. Specifically, these will be the parties who run a consensus protocol to validate the authenticity and validity of the shared resources—similarly to the role of nodes in the existing blockchain systems, such as Bitcoin [14].² We note that, in principle, the data owners may also perform the role of nodes.

1.1 Problem Definition and Motivation

For blockchain technology to serve as a backbone for the implementation of data cooperatives, their design and operational structure must follow the typical architecture of a blockchain system [7, 15, 16]. Specifically, this means that the data or resources pooled by the members should be included in the transactions of the underlying blockchain system. Then, they will be validated by consensus nodes before being added to the chain. At the same time, due to the community-based nature of the data cooperative's functioning, it is desirable to ensure the inclusivity of all members. For example, in a neighborhood watch system [11], community members come together to share news and information about their neighborhood as a way to improve security and increase awareness of issues happening in their community. Naturally, the inclusivity principle has to be reflected in the selection of nodes into the consensus committee such that even a newly joined node—potentially having a relatively low reputation—has a fair chance of being selected.

¹ DID refers to an individual owning personal digital data relating to multiple elements of one's identity.

² We would like to refer an unfamiliar reader to these detailed surveys [15, 19] on blockchain systems.

1.2 Our Contributions

The main contribution of this work is a proposal for a reputation-based blockchain system architecture suitable for data cooperatives. The key points of our solution are listed below, and we discuss them in detail in Sect. 3.

- The overall design is based on the reputation-based construction of Kleinrock et al. [17]. Specifically, we use their reputation-based lottery system for the selection of nodes into a consensus committee. This allows us to ensure inclusivity using *reputation fairness* [17], i.e., proportional representation (in the committee) of parties from all reputation tiers.
- Unlike [17], we do not use the Proof-of-Stake (PoS) chain as a fallback but rather rely on it as the main chain. The rationale is that transaction speed is not a bottleneck in the data cooperative scenario (as opposed to Fintech applications). At the same time, the majority of stakes assumption is more reasonable in our community-based environment, as compared to the super-majority³ assumption for the Proof-of-Reputation chain in [17].
- The above-mentioned simplification entails the following two aspects: First, one needs to map reputation to the stake, which can be done in several ways depending on a specific application. For simplicity in this work, we propose a straightforward mapping to the weighted stake, as described in Sect. 3.5. Second, we are now able to use an off-the-shelf PoS blockchain system in our architecture, which provides flexibility to our design.
- Specifically, we propose to use the Snow White PoS system [18] due to its very convenient modular property: it is secure as long as the honest parties constitute a majority in the committee. Another useful property of this system (for our community-based environment) is the so-called “sleepy consensus”, which allows parties to join and leave the consensus protocol as needed. In Sect. 4, we provide a sketch of a security proof for the proposed system under assumptions that the underlying reputation system satisfies a certain *feasibility* [17] property and the honest parties hold a majority of total reputation in the system.

1.3 Organization of the Paper

The remaining sections of this paper are organized as follows: Section 2 reviews the related literature and discusses the differences with our work. Section 3 describes the proposed system architecture and framework in detail. Section 4 presents a sketch of the security analysis. In Sect. 5, we discuss preliminary simulation results concerning the committee selection. We conclude the paper in Sect. 6 by summarizing the key concepts and ideas of this paper.

³ That is an assumption that over a 2/3 fraction of the overall reputation is held by honest parties.

2 Literature Review

2.1 Data Cooperatives

Data cooperatives can exist in various forms depending on the nature of data or resources being gathered by their members. Salau et al. [11] developed a proof-of-concept blockchain-based data coop as a form of neighborhood watch where its members can share phishing information in a decentralized and timely manner. MIDATA⁴ is an existing data coop where members can contribute to medical research by granting controlled access to the use of their personal data for research purposes. HAT⁵ is another such platform where individuals have the right to the ownership of their personal data. Zyskind et al. [20] added privacy to the data cooperative solution such that different parties can run joint computation while protecting the data from unauthorized access. Other related works include, e.g., [21, 22] where media organizations pool their resources for better news coverage or facility sharing. However, these latter two projects differ from ours in that they are specific to a use case, centralized, and do not employ blockchain technology.

2.2 Blockchain Systems

The proof-of-stake (PoS) consensus [23] was introduced as an energy-efficient alternative to the proof-of-work (PoW) consensus mechanism. After that, other forms of consensus protocols have been developed, which capitalize on various forms of assets as a criterion for participating in the blockchain augmentation. For example, in the PoW-based systems, computational resources serve as an asset that gives the *miner* a chance of being selected to add a new block. In the PoS-based systems, the amount of assets possessed by a stakeholder determines their eligibility for being selected into a committee, which performs block validation or adds a new block [24]. Such assets can be of different types depending on the blockchain implementation: e.g., the total amount of coins, coin age, reputation, and others.

Two fundamental design challenges in PoS-based blockchain systems are the simulation of the leader election in a fair and randomized manner [25, 26] and lack of “fairness” with respect to less rich participants [27, 28]. The election process has to be randomized enough that an adversary cannot predict its outcome [29], while also secure enough to prevent the so-called “grinding” attack where an adversary can use computational resources to influence the leader selection process.

⁴ <https://www.midata.coop/en/home/>.

⁵ <https://www.hubofallthings.com/main/what-is-the-hat>.

A lot of Proof-of-Reputation (PoR) based blockchain systems follow the basic concept of PoS. For instance, the constructions [30, 31] replaced coin-based stake voting with reputation-based voting where a node with the highest reputation is selected as a leader while the other top 20% of high-reputation nodes are selected as validators, and the associated incentives are shared among the validators in proportion to their reputation values. In Repucoin [32], the authors designed a PoR system that is based on PoW where a miner's power is determined by its reputation, derived from the work they have put in over the life of the blockchain. Reputation has also been used as a factor in reward and penalty systems [33–35]. A hybrid of reputation and another resource has also been used in the existing literature [36, 37]. The work [27] uses reputation in blockchain-based e-commerce settings for privacy preservation.

Kleinrock et al. in their work [17], which is most closely related to ours, leveraged reputation to design a system where newly joined nodes in the system are given a fair chance of participation in the protocol—a notion they refer to as *reputation fairness*. Differently from our work, they use the system of two blockchains: the main Proof-of-Reputation-based chain, and the fallback chain, which is based on Proof-of-Stake.

Distribution of Stakeholders. In blockchain systems, one or more nodes are selected to perform some system-specific tasks such as block generation, as in PoW-based systems or consensus committee selection in PoS-based systems. This selection can be done based on various factors such as reputation [30, 37], amount of coins [25], coin age [23], computational power [14], and others. In the blockchain systems that require a formation of the consensus committee, some form of randomness is employed to prevent adversaries from predicting the selected nodes [29].

When the committee is finally formed, the statistical distribution of the selected nodes is either triangular, exponential, exponential power, or some other form of probability distribution [38]. Specifically, in [38, 39], the authors compared selection into committees using triangular, exponential, and normal selections. Their results showed that when an exponential distribution is used, predominantly the nodes with the highest reputation are selected into the committee, while the triangular selection distribution gives a better chance to nodes with a lower reputation. The ReCon system [40]—another work, which is closely similar to ours—applies an external reputation ranking as its input for selecting nodes for a committee. This approach is different from ours in that the reputation system is integrated into our design and our selection process is tier-based, which grants a better chance to newly joined nodes with low reputations.

2.3 Reputation Systems: Formalization

In this subsection, we generally follow the presentation by Kleinrock et al. [17]. We use the same formalization and security definitions for reputation systems as those from [17] (which in turn relies on [41]).

A reputation system Rep for n parties (or peers) from a set $\mathcal{P} = \{P_1, \dots, P_n\}$ is a family of probability distributions (parametrized by n) over binary reputation vectors of length n , i.e., vectors of the type $(h_1, \dots, h_m) \in \{0, 1\}^n$. Each h_i is an indicator bit such that $h_i = 1$, if P_i is honest, and $h_i = 0$, otherwise. A reputation system as described above is called *static*.

Here, our focus is on such a setting where each h_i corresponds to the outcome of an independent indicator random variable, H_i . This means that whether or not a party P_i behaves honestly does not depend on what other parties do. In this case, a static reputation system can be defined as follows: $Rep = (R_1, \dots, R_n) \in [0, 1]^n$, where R_i is the probability that the party R_i will play honestly. We then say that R_i is the reputation value (or simply *reputation*) of a party P_i . Such a construction is called a *correlation-free* reputation system. In this work, we will focus on this type of reputation system.

The reputation system dictates the adversary's corruption capabilities. That is, a static reputation-bounded adversary for a reputation system Rep for n parties (Rep -adversary for short), corrupts the set of parties at the beginning of the protocol according to Rep , and this set remains fixed.

Definition 1 For a (possibly probabilistic) algorithm \mathbf{A} for sampling a subset of parties from \mathcal{P} , and a Rep -adversary A , we say that a reputation system Rep is (ϵ, \mathbf{A}) -feasible for A if with overwhelming probability—taken over the coins associated with the distribution of Rep and the coins of A and \mathbf{A} —the algorithm \mathbf{A} outputs a set of parties such that at most a $1/2 - \epsilon$ fraction of these parties is corrupted by A .

Definition 2 A reputation system is ϵ -feasible for Rep -adversary A , if there exists a probabilistic polynomial time sampling algorithm \mathbf{A} such that Rep is (ϵ, \mathbf{A}) -feasible for A .

2.4 Multi-Tier Selection Protocol

We present the multi-tier lottery by Kleinrock et al. [17], which we use for selection of the consensus committee. In this section, we generally follow the presentation of [17]. The tiers can be $2, 3, \dots, m$. For example, given a small $\delta > 0$, the first tier will have reputation values between $\frac{m-1}{m} + \delta$ and 1, while the second tier will have reputation values between $\frac{m-2}{m} + \delta$ and $\frac{m-3}{m} + \delta$, and so on. This is done to avoid the famous “rich getting richer” problem, which is common to stake-based consensus protocols, and to afford nodes with a fairly good but not top-tier reputation a chance of being part of the consensus committee.

Note that the parameter δ defines the difference in probabilities for each tier to be selected, i.e., a node in the first tier has δ times a chance of being chosen compared to a node in the second tier and so on. A vector R represents the reputation values of the potential stakeholders, and specifically R_j represents the reputation values of the potential stakeholders in partition j . The parameter ϵ is the committee size factor, which influences the total number of stakeholders that can be selected into different tiers. The value T represents the total number of tiers to be partitioned, and c_i represents the ratio between two successive tiers which is initialized to 1.

Let T_1, \dots, T_m be a partition of the consensus committee candidate nodes into m tiers as described above with nodes in partition T_1 having the highest reputation values and, γ_i be the total number of stakeholders selected in tier T_i . Let **elect_cmt** be the selection protocol that selects l_i parties from each of the T_i partitions. The multi-tier lottery protocol, which is shown in Fig. 2, is taken from [17].

elect_cmt($T, R, m, \epsilon, \delta, (c_1, \dots, c_m = 1)$)

1. Divide the reputation parties into m tiers T_1, \dots, T_m as follows:
 For $i = 0, \dots, m - 1$ define T_{m-i} to be the set of nodes in T with reputation $R_j \in \left(\frac{i}{m} + \delta, \frac{i+1}{m} + \delta \right)$
2. Initialize $\bar{T}_i := \emptyset$ for each $i \in [m]$
3. For each $i = 1, \dots, m$: Let $\gamma_i := |T_i|$
4. For each $i = 1, \dots, m - 1$: Let

$$\ell_i = \frac{\sum_{j=1}^i \gamma_j}{\sum_{j=1}^m \prod_{q=1}^j c_q} \frac{\gamma_{i+1} \prod_{j=1}^m c_j - \gamma_i \prod_{j=i+1}^m c_j}{\gamma_{i+1} \gamma_i} \log^{1+\epsilon} N \quad (1)$$

and

$$\ell_m = \frac{\sum_{j=1}^m \gamma_j}{\sum_{j=1}^m \prod_{q=1}^j c_q} \frac{1}{\gamma_m} \log^{1+\epsilon} N \quad (2)$$

5. For each $i = 1, \dots, m - 1$: do:
 if $|\cup_{j=1}^i T_j| \geq \ell_i$:
 $\text{Rand}(\cup_{j=1}^i T_j, \lceil \ell_i \rceil)$
 For each $j \in [i]$ compute $Q_{i,j} := Q_i \cap \bar{T}_j$ and update $\bar{T}_j \cup (Q_i \cap T_j)$
 else:
 Reset the protocol and run again
6. Output $P_{sel} := \cup_{j=1}^m \bar{T}_j$

Fig. 2 Multi-Tier lottery protocol **elect_cmt** ([17])

3 Description of the Proposed Blockchain System

First, let us state the design requirements for the proposed system.

3.1 Design Requirements for Data Cooperative

Data Cooperative Properties

1. The infrastructure must be decentralized, without the presence of a sole centralized authority (as such authority may have its obvious limitations with regard to scalability, security, fairness, and objectivity).
2. There should be a common and immutable log of activities.
3. How about the following: The infrastructure should support as many participants as possible, and it should be publicly available.
4. The committee selection protocol must be fair to newly joined nodes, although high-reputation nodes must have a clear advantage. (This would provide incentives for productive behavior).
5. The consensus protocol should tolerate malicious behavior by a coalition of parties having a minority of the total reputation.
6. The blockchain with a transaction ledger satisfies the standard *liveness* and *consistency* properties [18] with overwhelming probability in the security parameter κ , that is the number of blocks after which a transaction is regarded as a permanent part of the blockchain history.

Reputation System Properties

1. Reputation must be earned. It cannot be purchased, traded, or spent.
2. Reputation of a node is the aggregation of all reputation adjustments to the current round for that node.
3. Peers should not be responsible for directly computing their respective reputation values, and neither should they be able to modify these values.
4. Reputation values must be stored on the blockchain and be verifiable by all peers.

3.2 Architecture Overview

In this section, we describe the architecture of our work. As shown in Fig. 3, at the highest (application) layer, there can be multiple distributed applications (DApps) deployed on our system. Consider for instance a data sharing scenario of Fig. 4, where a member of the data cooperative shares a piece of news information on the platform. Then, on receipt of the news as a transaction, all nodes interested in participating in the validation process submit a token that confirms their interest.

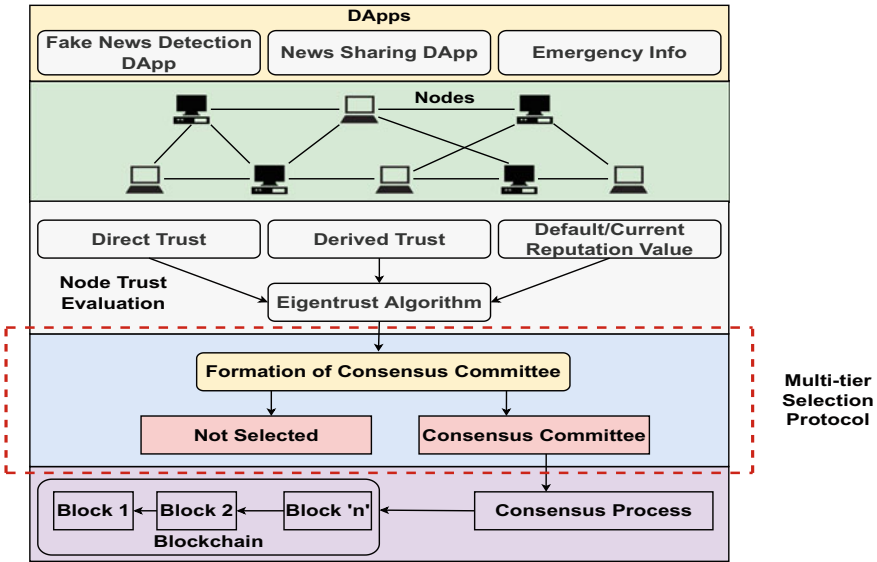


Fig. 3 Proposed system architecture. There can be multiple DApps deployed on the system, which users can employ to interact with the blockchain. The EigenTrust [42] reputation system computes the reputation of each nodes participating in validation, while the multi-tier selection protocol then selects the consensus committee based on the reputation tiers. The committee validates transactions and adds blocks to the blockchain

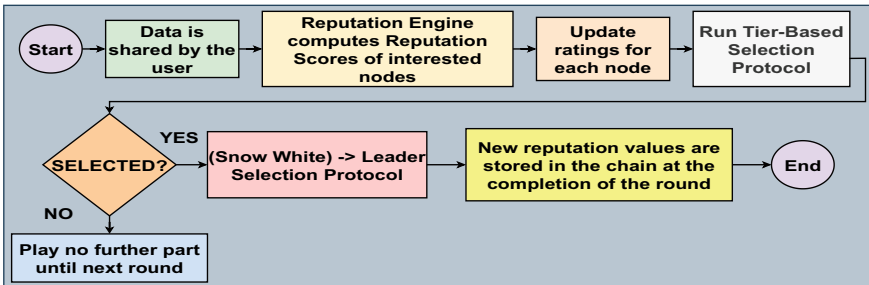


Fig. 4 Event flow diagram for the proposed system. When a member of the data cooperative shares a piece of data on the platform, all nodes interested in participating in the validation process submit a token that confirms their interest. The reputation system then retrieves the current reputation values of each of the nodes and partitions the nodes into different tiers based on our selection protocol

The reputation system then retrieves the current reputation values of each of the nodes and partitions the nodes into different tiers based on their reputation values. Nodes with a reputation value of zero are ignored (since a node’s reputation can only drop to zero if it is malicious). The tier-based selection protocol is executed and the total number of nodes required for the consensus is selected based on the protocol parameter ϵ (this parameter was introduced in Sect. 2.4). For nodes not chosen, there

is no further part to play until the next epoch, while the selected stakeholders go on to run a secure multi-party coin-flipping protocol to elect a block proposer and the input endorsers as described in Snow White [18]. At the end of the epoch, the reputation value of each of the participating stakeholders is updated and stored on the chain.

In Snow White [18], parties can join freely by creating coins, and making/receiving payments, with stake shifting over time. The protocol execution runs in epochs and a set of random stakeholders is selected to form a committee that will run a secure multi-party coin-flipping protocol to determine the set of leaders for the epoch. The Snow White protocol supports an application-specific selection mechanism. In our case, we will use protocol `elect_cmt()`, which is described in detail in Sect. 2.4. This protocol takes input in the form of parties' reputation (stored on the blockchain) and outputs a set of parties, which will form the consensus committee.

Our system uses the Snow White protocol and hence it inherits all the features and security, which this protocol offers, with an exception of formation of the committee from which the block proposers are chosen. Note, however, that our committee selection protocol can be used with any system that requires selecting nodes and requires an honest majority of reputation. In our work, we replace the sampling weighted by stake with the one weighted by reputation since in our data cooperative scenario, the reputation of the members is critical for functioning of the system. (See Sect. 3.5 for details).

3.3 *Choice of Blockchain System*

Snow White [18] is a PoS-based blockchain system which has a formal security proof. Compared to other PoS-based systems of this type, such as the Ouroboros family [25, 43] and Algorand [44], Snow White has a convenient modular feature that its security proof holds as long as the committee has an honest majority. This provides a capacity to support a wide range of applications via application-specific committee selection algorithms. Since our data cooperatives are expected to support a number of distributed applications (see Sect. 3.2 for discussion on some of them), the above-mentioned property is very useful in our setting. Moreover, Snow White uses the sleepy consensus protocol [45], which is an asynchronous protocol that enables the committee reconfiguration as parties join and leave the protocol execution at will. This feature makes the protocol robust against unstable network conditions where disconnections and message delays are possible. Snow White promotes participation fairness, inclusivity, and overall system sustainability, which matches our overall inclusivity requirements for the design of data cooperatives.

3.4 Choice of Reputation System

For this work, we adopted the EigenTrust reputation system [42]. There are a lot of reputation systems for trust management in P2P systems, such as SuperTrust [46], Regret [47], Fire [48], Core [49], PeerTrust [50], and Travos [51]. We refer the reader to the work by Vavilis et al. [52] for a survey. In particular, the EigenTrust algorithm works by identifying and isolating malicious peers, thereby mitigating their impact on the performance of a P2P system [53, 54] through computation of global reputation scores of participating peers. EigenTrust is a popular system, which was studied and used in a wide variety of works, e.g., [10, 42, 55, 56] to mention a few. As compared to the above-mentioned reputation systems, EigenTrust ensures protection for new users from being penalized for their status. Also, the users are not directly responsible for calculating their own values or for the reputation values of others [52]. These features match our design goals for data cooperative, in particular, they ensure that low-reputation newly joining peers have a fair representation in the system decision-making process. Note that such newly joined nodes are clearly differentiated from the peers who have a low reputation due to punishments received. This is because if new peers are treated in the same manner as existing users with a bad reputation, they may never be selected for system-specific tasks like transaction validation, and therefore, they cannot improve on their reputation [42].

In EigenTrust, the global trust value (which is the reputation in our context) is computed by combining a direct trust value between a user and other users it had direct transactions with, and a recommended trust value, which is based on transitive trust [59].

The direct trust between users is computed as follows:

$$c_{i,j} = \begin{cases} \frac{\max(s_{i,j},0)}{\sum_j \max(s_{i,j},0)}, & \text{if } \sum_j \max(s_{i,j},0) \neq 0; \\ p_i = 1/|P|, & \text{otherwise,} \end{cases} \quad (3)$$

where $c_{i,j}$ denotes the direct trust between users i and j , $s_{i,j}$ is the difference in the satisfactory and unsatisfactory transactions between users i and j , which is computed as $s_{i,j} = sat_{i,j} - unsat_{i,j}$ (see [42] for details). The value p_i represents a case where peer i may be new and does not trust any other peer, then it will have to choose the pre-trusted peers, and the set of such peers is denoted as P .

The derived trust is computed as follows:

$$c_{i,k} = \sum_j c_{i,j} c_{j,k}, \quad (4)$$

which is based on the direct trust computed in (3).

We use an alternative version of the global trust computation from [56], incorporating an additive increase and a multiplicative decrease penalty component to it in order to provide better robustness to the reputation system. Specifically, in a distributed setting, there is a possibility that malicious users collude to assign arbitrarily

high trust values to each other and arbitrarily low trust values to honest peers. This can be addressed by introducing a proliferation parameter a , thereby recalculating the current reputation of each peer as follows:

$$t_i = (1 - a)(c_{1,i}t_1 + c_{2,i}t_2 + \dots + c_{n,i}t_n) + ap_i, \quad (5)$$

where a is a constant ($a \leq 1$).

3.5 Reputation as Weighted-Stake

For the selection of the consensus committee, we use the tier-based lottery of [17]. This protocol is described in Sect. 2.4. Specifically, in the “Formation of Consensus Committee” phase of Fig. 3, the consensus committee is formed using this lottery protocol.

Note that the parties are selected into the consensus committee based on their reputation values, but the underlying blockchain system is based on stake. Therefore, the reputation values of the stakeholders need to be mapped to a stake on the blockchain. For this, we use the following straightforward mapping, where s_i is the weighted stake of party i derived from its reputation R_i and $\sum_{k=1}^n R_k$ is the total reputation of all the parties in the system:

$$s_i = \frac{R_i}{\sum_{k=1}^n R_k}. \quad (6)$$

The reputation values and their use for committee selection are described in Sect. 2.4.

Finally, we note that the above approach for mapping reputation into stake is taken for the simplicity of our presentation. Different methods such the mapping may be used depending on specific applications.

4 Security Analysis

Let us present a sketch of the security analysis for the proposed system. First, let us informally describe the intuition behind our security argument.

As stated above, the security of the proposed system relies on the properties of the underlying reputation system and PoS-based blockchain engine. Specifically, we assume that this system adequately reflects the parties’ reputation and that a majority of reputation always remains with the honest parties, i.e., those who follow the protocol. With the mapping described in the previous subsection, the majority of reputation directly translates into the majority of stake, and hence, the PoS engine works properly.

Let us now state our security results more formally. We omit some parameters, which are not essential in our setting. Denote the EigenTrust reputation system (described in Sect. 3.4) by Rep for short. In this discussion, we rely on notation from Sect. 2.3.

Let the protocol **elect_cmt** (described in Sect. 2.4) be used for selecting the parties into the committee—it implements the sampling algorithm **A** from Definitions 1 and 2. We denote the set of parties selected into the committee as P_{sel} .

The following theorem is easily adapted from Theorem 1 of [17].

Theorem 1 ([17]) *If the reputation system Rep is ϵ_f -feasible, for some constant $\epsilon_f \in (0, 1)$, for a static Rep -bounded adversary A , then A corrupts at most $1/2 - \epsilon_\delta$ fraction of the parties in P_{sel} , for some constant $\epsilon_\delta \in (0, 1)$, with overwhelming probability in the security parameter.*

The proof of the above theorem directly follows from that of Lemma 4 of [17].

Let the blockchain system parameters be chosen as prescribed by the Snow White protocol [18]. Specifically, Appendices C.5-C.6 and F.1 of [18] described the admissible parameters, including the probabilities for honest and dishonest parties to be selected to the committee in order for the blockchain system to work properly, i.e., to ensure chain growth, chain quality, and consistency, as defined in Appendices B.1-B.3 of [18]. In particular, these admissible parameters are to ensure that P_{sel} has a majority of honest parties. It is easy to see that the value ϵ_δ can always be chosen to ensure that the parameters of the Snow White protocol are indeed admissible.

Theorem 2 ([18]) *There exists a parameter $\epsilon_\delta \in (0, 1)$ such that if the adversary corrupts at most $1/2 - \epsilon_\delta$ fraction of the parties in P_{sel} , the Snow White protocol provides chain growth, chain quality, and consistency properties.*

The proof of the above theorem follows immediately from that of Theorem 1 in [18] by setting the parameter ϵ_δ , which satisfies the admissibility conditions formulated in Appendix C.6 of [18].

Then, the next result follows immediately from the above two theorems, taking into account the mapping of reputation to stake as described in Sect. 3.5.

Corollary 1 *If the EigenTrust reputation system described in Sect. 3.4 is ϵ_f -feasible, with a constant $\epsilon_f \in (0, 1)$ chosen according to Theorem 1, for a static Rep -bounded adversary A , then the blockchain system proposed in Sect. 3 provides chain growth, chain quality, and consistency properties.*

As pointed out in Appendix G of [17], the above result directly translates to the case of dynamic reputation systems under an assumption that reputation vectors in different epochs are independent. A formal treatment of the dynamic case when the above assumption is removed is left as an open question for our future work.

5 Experimental Results and Discussion

In this section, we present preliminary simulation results concerning our implementation of the tier-based selection protocol **elect_cmt**. As concluded in [39, 40], the triangular selection distribution gives a comparatively higher chance to newly joined low-reputation parties to be chosen, while the exponential selection distribution mainly selects high-reputation nodes while giving little chance of selection to low-reputation nodes.

5.1 Datasets

First, let us discuss the datasets used in our experiments. In order to study the performance of the **elect_cmt** protocol, we used real-life datasets of reviews from Amazon.com [57] and Ebay.com [58]. The Amazon.com dataset [57] (we will call it Dataset 1 for short) has user star ratings on a scale of 1 to 5 with 5 stars being the highest. The Ebay.com dataset [58] has two sets of review ratings in it. One set (we call it Dataset 2a) has the same 5-star rating as in Dataset 1. Another set has ratings on a scale of $(-1, 0, 1)$ with 1 being the highest (we call it Dataset 2b). In order to be able to use these datasets for our work, we equate having a 5-star rating with a reputation value in the interval of $(0.8, 1.0]$, a 4-star rating with $(0.6, 0.8]$, and so on until a 1-star rating with $(0, 0.2]$, while those with a reputation value of 0 are ignored in datasets 1 and 2a. Also, for dataset 2b, we equate a rating of 1 with reputation values of $(0.66, 1]$, a rating of 0 with $(0.33, 0.66]$ and a rating of -1 with $(0, 0.33]$, again, ignoring peers with a reputation value of 0.

5.2 Experiments and Discussion

The main objective of the experiments we carried out was to confirm an inclusive nature of the tier-based selection approach when compared to the ranking-based approach used in [30, 41]. In the latter case, the formation of a consensus committee is done by ranking the potential stakeholders in order of their reputation and then choosing the stakeholders with the highest reputation to join the committee until the required number is reached.

We implemented the protocol **elect_cmt**, ran 100 instances of the protocol, averaged the result, and observed the distribution of selected users by tiers according to their reputation versus the ranking-based approach described above.

We chose the number of tiers T as 3 and 5 in order to match the respective datasets. Also, we used different tier ratios c , which were set to 2 and 3. This parameter governs the difference in the number of parties between tiers. Roughly speaking, when $c = 2$, we may expect twice as many peers in Tier 1 as compared to Tier 2, and so on. The

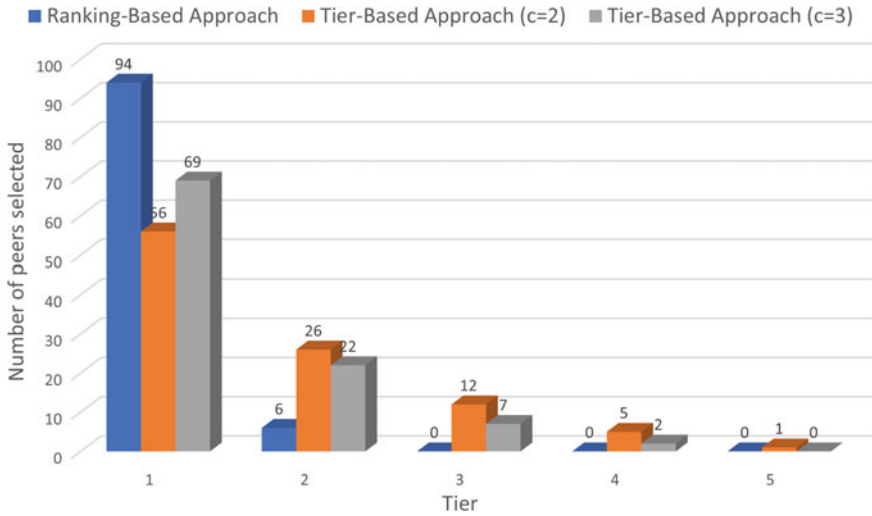


Fig. 5 Number of peers chosen to the committee from different tiers, according to tier-based versus ranking-based approach, for dataset 1

parameters used for the experiment are summarized in Table 1. These parameters were introduced in Sect. 2.4.

In the first experiment, we performed the comparison for dataset 1. The result is shown in Fig. 5. As we can see, for the ranking-based approach, parties in tiers 3, 4 and 5 were completely ignored, while in the tier-based approach, we had a fair representation of the peers in the lower tiers. This demonstrates that the inclusivity, a core requirement in our data cooperative design, as achieved when using the tier-based approach.

In the second experiment with the result as shown in Fig. 6, we compared the above mentioned approaches for dataset 2a. We can see that the distribution of parties into tiers is close to that from the previous experiment, as expected.

Finally, in the third experiment, we compared the tier-based and rank-based approaches for the case of 3 tiers using dataset 2b. The result is shown in Fig. 7. Again, we can observe that both selection mechanisms have the same number of peers selected from Tier 1 but the tier-based approach also chooses peers in the lowest third tier, which was ignored by the ranking-based approach. This is because the total number of peer candidates in Tier 1 (those with a reputation value of (0.66, 1.0], which is equivalent to a 1-star rating) is 32, and they are normally all selected to fill the 100 required places in the committee. However, regarding the peers selected from Tiers 2 and 3, we can see that the tier-based approach “fairly” considered 7 peers from Tier 3 (which would be the newly joined low-reputation peers in the data coop setting), while the ranking-based approach completely neglected them. Notice also, that in the tier-based selection approach, the ratio between the peers selected from Tier 2 and Tier 3 is far larger than the 2 or 3, which was observed in the previous

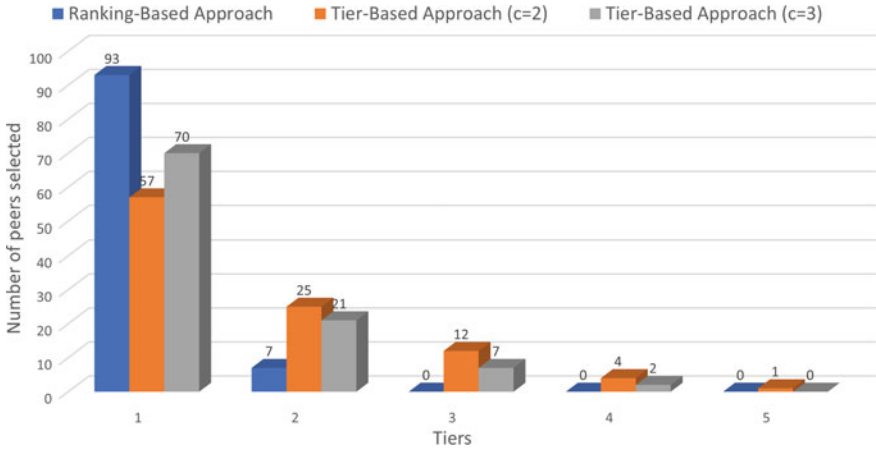


Fig. 6 Number of peers chosen for the committee from different tiers, according to tier-based versus ranking-based approach, for dataset 2a

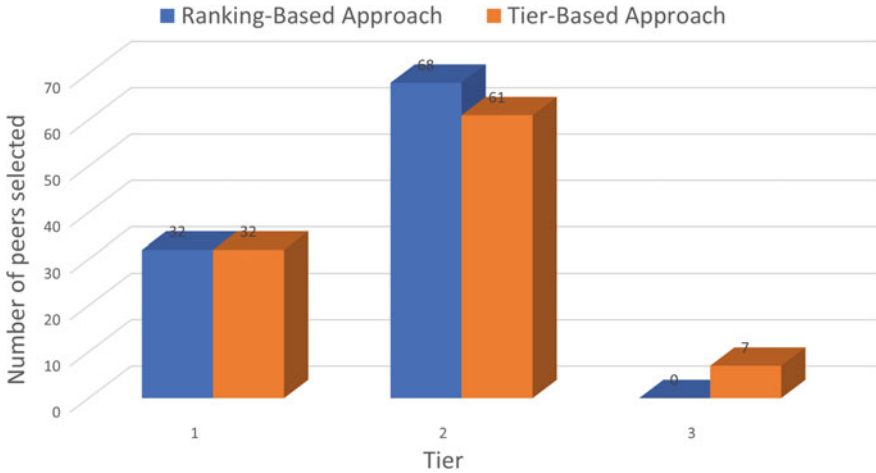


Fig. 7 Number of peers chosen to the committee from different tiers, according to tier-based versus ranking-based approach, for dataset 2b

experiments. This is because, according to the formula for the selection ratio c_j shown in the last entry of Table 1, the total number of candidates with reputation values in $(0.33, 0.66]$ (Tier 2) is far greater than the candidates with reputation values in $(0, 0.33]$ (Tier 3).

Table 1 Experiment parameters

Parameter	Value
Total number of nodes (N)	1000
Number of tiers (m)	{3,5}
Parameter δ	0.01
Committee size factor (ϵ)	2
Committee size(n)	$\log^{1+\epsilon}(N)$
Tier ratio ($c \geq 1$)	{2,3}
Tiers selection ratio (c_j)	$\max\{c, c \cdot \frac{ T_j }{ T_{j+1} }\}$

6 Conclusion

Data cooperatives are an important mechanism for the new data economy and also for establishing trust between their members, which is crucial for their operation. Reputation is an important measure of the trustworthiness of entities, which can effectively complement financial assets in the decentralized scenario. In addition, it will lead to more democratic governance in the distributed data management systems, by giving more power to law-abiding and socially active individuals while also providing an incentive to others to behave as such. The Proof-of-Stake and Proof-of-Reputation systems seem to be particularly promising examples of the next-generation blockchain technologies, which will enable the effective functioning of data cooperatives.

This work is the first step towards the design and implementation of decentralized data cooperatives based on Proof-of-Reputation blockchains. In our subsequent work, we plan to equip the existing system with data encryption and privacy-preserving mechanisms such as zero-knowledge proofs in order to enable fine-grained control of digital assets by their owners.

Acknowledgements We thank the National Security Agency for the partial support through grants H98230-20-1-0329, H98230-20-1-0403, H98230-20-1-0414, and H98230-21-1-0262. We are grateful to Stefanos Leonardos and the anonymous reviewers for their helpful comments.

References

1. World Economic Forum. (2014). “Rethinking Personal Data: A New Lens for Strengthening Trust.” <http://reports.weforum.org/rethinking-personal-data>.
2. Kearney, A. T. (2014). Rethinking personal data: A new lens for strengthening trust. In *World Economic Forum*. Retrieved November (Vol. 1).
3. Pentland, A. (2020). Building the new economy: What we need and how to get there. In *Building the New Economy*.
4. Hardjono, T., & Pentland, A. (2019). Data cooperatives: Towards a foundation for decentralized personal data management. arXiv preprint [arXiv:1905.08819](https://arxiv.org/abs/1905.08819).

5. Ada Lovelace Institute. (2021). Data cooperatives. In Chapter two from Exploring legal mechanisms for data stewardship—a joint publication with the AI Council. Available at <https://www.adalovelaceinstitute.org/feature/data-cooperatives/>.
6. Data Co-Ops Workshop. (2019). Executive summary of a December 22, 2019 workshop hosted at the Hebrew University of Jerusalem. *The Federmann Cyber Security Research Center*. Available at https://csrcl.huji.ac.il/sites/default/files/csrcl/files/data_co_ops_summary.pdf.
7. Zyskind, G., Nathan, O., & Pentland, A. (2015). Decentralizing privacy: Using blockchain to protect personal data. *IEEE Security and Privacy Workshops, 2015*, 180–184.
8. Salau, A., Dantu, R., Morozov, K., Upadhyay, K., & Badruddoja, S. (2022). Towards a threat model and security analysis for data cooperatives. In *Proceedings of the 19th International Conference on Security and Cryptography*. ISBN 978-989-758-590-6, ISSN 2184-7711, pp. 707–713.
9. Schaub, A., Bazin, R., Hasan, O., & Brunie, L. (2016). A trustless privacy-preserving reputation system. In *IFIP International Conference on ICT Systems Security and Privacy Protection* (pp. 398–411). Springer.
10. Gao, S., Yu, T., Zhu, J., & Cai, W. (2019). T-PBFT: An EigenTrust-based practical Byzantine fault tolerance consensus algorithm. *China Communications*, 16, 111–123.
11. Salau, A., Dantu, R., & Upadhyay, K. (2021). Data Cooperatives for Neighborhood Watch. *IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2021*, 1–9.
12. Gupta, D. (2022). Decentralized identity using blockchain. Available online at <https://venturebeat.com/2022/03/05/decentralized-identity-using-blockchain/>.
13. Heister, S., & Yuthas, K. (2021). How blockchain and AI enable personal data privacy and support cybersecurity. In T. M. Fernández-Caramés & P. Fraga-Lamas (Eds.), *Advances in the Convergence of Blockchain and Artificial Intelligence*. IntechOpen.
14. Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, 21260.
15. Ferdous, M. S., Chowdhury, M. J. M., Hoque, M. A., & Colman, A. (2020). Blockchain consensus algorithms: A survey. arXiv preprint [arXiv:2001.07091](https://arxiv.org/abs/2001.07091).
16. Magyar, G. (2017). Blockchain: Solving the privacy and research availability tradeoff for EHR data: A new disruptive technology in health data management. In *2017 IEEE 30th Neumann Colloquium (NC)* (pp. 000135–000140).
17. Kleinrock, L., Ostrovsky, R., & Zikas, V. (2020). Proof-of-reputation blockchain with Nakamoto fallback. In *International Conference on Cryptology in India* (pp. 16–38). Springer. Full version: Cryptology ePrint Archive, Paper 2020/381. <https://eprint.iacr.org/2020/381.pdf>.
18. Daian, P., Pass, R., & Shi, E. (2019). Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *International Conference on Financial Cryptography and Data Security* (pp. 23–41), Springer. Full version: Cryptology ePrint Archive, Paper 2016/919. <https://eprint.iacr.org/2016/919.pdf>.
19. Wang, W., Hoang, D. T., Hu, P., Xiong, Z., Niyato, D., Wang, P., & Kim, D. I. (2019). A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, 7, 22328–22370.
20. Zyskind, G., Nathan, O., & Pentland, A. (2015). Enigma: Decentralized computation platform with guaranteed privacy. ArXiv abs/1506.03471.
21. Banyan Project. (2021). Our product: Trustworthy news and information that stir civic engagement. Accessed January 30, 2021, from <https://banyanproject.coop/>.
22. The Associated Press. (2021). Our mission is to inform the world. Accessed January 30, 2021, from <https://www.ap.org/en-us/>.
23. King, S., & Nadal, S. (2012). Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper. 19(1).
24. Leonardos, S., Reijnders, D., & Piliouras, G. (2020). Weighted voting on the blockchain: Improving consensus in proof of stake protocols. *International Journal of Network Management*, 30(5), e2093.
25. Kiayias, A., Russell, A., David, B., & Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz, & H. Shacham (Eds.), *Advances in Cryptology—CRYPTO 2017*. CRYPTO 2017. Lecture Notes in Computer Science, vol 10401, Springer.

26. Han, X., Yuan, Y., & Wang, F.-Y. (2019). A fair blockchain based on proof of credit. *IEEE Transactions on Computational Social Systems*, 6(5), 922–931.
27. Zou, J., Ye, B., Qu, L., Wang, Y., Orgun, M. A., & Li, L. (2018). A proof-of-trust consensus protocol for enhancing accountability in crowdsourcing services. *IEEE Transactions on Services Computing*, 12(3), 429–445.
28. Narayanan, A., Bonneau, J., Felten, E., Miller, A., & Goldfeder, S. (2019). Bitcoin and cryptocurrency technologies. Curso elaborado pela.
29. Wang, Q., Xu, M., Li, X., & Qian, H. (2020). Revisiting the fairness and randomness of delegated proof of stake consensus algorithm. In *2020 IEEE Intl Conf on Parallel and Distributed Processing with Applications, Big Data and Cloud Computing, Sustainable Computing and Communications, Social Computing and Networking (ISPA/BDCLOUD/SocialCom/SustainCom)* (pp. 305–312). <https://doi.org/10.1109/ISPA-BDCLOUD-SocialCom-SustainCom51426.2020.00064>.
30. Zhuang, Q., Liu, Y., Chen, L., & Ai, Z. (2019). Proof of reputation: A reputation-based consensus protocol for blockchain based systems. In *Proceedings of the 2019 International Electronics Communication Conference* (pp. 131–138).
31. Do, T., Nguyen, T., & Pham, H. (2019). Delegated proof of reputation: A novel blockchain consensus. In *Proceedings of the 2019 International Electronics Communication Conference* (pp. 90–98).
32. Yu, J., Kozhaya, D., Decouchant, J., & Esteves-Verissimo, P. (2019). Repucoin: Your reputation is your power. *IEEE Transactions on Computers*, 68(8), 1225–1237.
33. Wang, E. K., Liang, Z., Chen, C. M., Kumari, S., & Khan, M. K. (2020). PoRX: A reputation incentive scheme for blockchain consensus of IIoT. *Future Generation Computer Systems*, 102, 140–151.
34. Gai, F., Wang, B., Deng, W., & Peng, W. (2018). Proof of reputation: A reputation-based consensus protocol for peer-to-peer network. In *International Conference on Database Systems for Advanced Applications* (pp. 666–681). Springer.
35. Bou Abdo, J., El Sibai, R., & Demerjian, J. (2021). Permissionless proof-of-reputation-X: A hybrid reputation-based consensus algorithm for permissionless blockchains. *Transactions on Emerging Telecommunications Technologies*, 32(1), e4148.
36. Zaccagni, Z., & Dantu, R. (2020). Proof of review (PoR): A new consensus protocol for deriving trustworthiness of reputation through reviews. *IACR Cryptol. ePrint Arch.*, 2020, 475.
37. Larangeira, M. (2021). Reputation at stake! A trust layer over decentralized ledger for multi-party computation and reputation-fair lottery. Cryptology ePrint Archive.
38. Bugday, A., Ozsoy, A., Öztaner, S. M., & Sever, H. (2019). Creating consensus group using online learning based reputation in blockchain networks. *Pervasive and Mobile Computing*, 59, 101056.
39. Biryukov, A., Feher, D., & Khovratovich, D. (2017). Guru: Universal reputation module for distributed consensus protocols. University of Luxembourg.
40. Biryukov, A., & Feher, D. (2020). ReCon: Sybil-resistant consensus from reputation. *Pervasive and Mobile Computing*, 61, 101109. Princeton University Press.
41. Asharov G., Lindell Y., & Zarusim H. (2013). Fair and efficient secure multiparty computation with reputation systems. In K. Sako, P. Sarkar (Eds.), *Advances in Cryptology—ASIACRYPT 2013*. ASIACRYPT 2013. Lecture Notes in Computer Science, vol 8270, Springer.
42. Kamvar, S. D., Schlosser, M. T., & Garcia-Molina, H. (2003). The eigentrust algorithm for reputation management in p2p networks. In *Proceedings of the 12th International Conference on World Wide Web* (pp. 640–651).
43. David, B., Gaži, P., Kiayias, A., & Russell, A. (2018). Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (pp. 66–98). Springer.
44. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. (2017). Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (pp. 51–68).

45. Pass, R., & Shi, E. (2017). The sleepy model of consensus. In *International Conference on the Theory and Application of Cryptology and Information Security* (pp. 380–409). Springer.
46. Dimitriou, T., Karame, G., Christou, I. T. (2008). SuperTrust—a secure and efficient framework for handling trust in super peer networks. In *Distributed Computing and Networking, LNCS* (vol. 4904, pp. 350–362). Springer.
47. Sabater, J., & Sierra, C. (2002). Reputation and social network analysis in multi-agent systems. In *Proceedings of International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 475–482). ACM.
48. Huynh, T. D., Jennings, N. R., & Shadbolt, N. R. (2006). An integrated trust and reputation model for open multi-agent systems. *Autonomous Agents and Multi-Agent Systems*, 13, 119–154.
49. Michiardi, P., Molva, R. (2002). Core: A collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks. In *Advanced Communications and Multimedia Security, IFIP AICT* (pp. 107–121). Kluwer Academic Publishers.
50. Xiong, L., & Liu, L. (2004). PeerTrust: supporting reputation-based trust for peer-to-peer electronic communities, *Transactions on Knowledge and Data Engineering*, 16, 843–857.
51. Teacy, W., Patel, J., Jennings, N., & Luck, M. (2006). Travos: trust and reputation in the context of inaccurate information sources. *Autonomous Agents and Multi-agent Systems*, 12, 183–198.
52. Vavilis, S., Petković, M., & Zannone, N. (2014). A reference model for reputation systems. *Decision Support Systems*, 61, 147–154.
53. Garcia-Retuerta, D., Casado-Vara, R., Valdeolmillos, D., & Corchado, J. M. (2021). A reputation score proposal for online video platforms. In G. Marreiros, F.S. Melo, N. Lau, H. Lopes Cardoso, & L. P. Reis (Eds.), *Progress in Artificial Intelligence. EPIA 2021. Lecture Notes in Computer Science*, vol. 12981, Springer.
54. Lu, K., Wang, J., Xie, L., Zhen, Q., & Li, M. (2016). An eigentrust-based hybrid trust model in P2P file sharing networks. *Procedia Computer Science*, 94, 366–371.
55. Zoë, A., Robert, M., & Serge, P. (2005). A non-manipulable trust system based on EigenTrust. *SIGecom Exch*, 5(4), 21–30.
56. Heba, A. K. (2015). HonestPeer. *Journal of King Saud University Computer Information Science*, 27(3), 315–322.
57. McAuley, J., Targett, C., Shi, Q., & Van Den Hengel, A. (2015). Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval* (pp. 43–52). Available at <https://jmcauley.ucsd.edu/data/amazon/>.
58. eBay Reviews and Guides. (2022). <https://www.kaggle.com/datasets/wojtekbionicki/ebay-reviews>, last viewed on May 28, 2022.
59. Audun, J., Elizabeth, G., & Michael, K. (2006). Simplification and analysis of transitive trust networks. *Web Intelligence and Agent Systems*, 4(2), 139–161.