Navigating decentralized finance (DeFi) risks and challenges through user-centric solutions

Mahsa Moosavi

A Thesis
In the Concordia Institute for
Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy (Information and Systems Engineering)
at Concordia University
Montréal, Québec, Canada

July 7, 2025

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: **Mahsa Moosavi**

Entitled: **Navigating decentralized finance (DeFi) risks and challenges through user-centric solutions**

and submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy (Information and Systems Engineering)

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

Luiz A.C. Lopes, Ph.D.	Chair
Majid Khabbazian, Ph.D.	External Examiner
Amr Youssef, Ph.D.	Examiner
Gengrui (Edward) Zhang, Ph.D.	Examiner
Tse-Hsun Chen, Ph.D.	Arms-Length Examiner
Jeremy Clark, Ph.D.	Supervisor

22 Apr 2025 *Mourad Debbabi, Ph.D., Dean (GCS)*

Abstract

Navigating decentralized finance (DeFi) risks and challenges through user-centric solutions

Mahsa Moosavi

Concordia University, 2025

This dissertation explores the evolving landscape of decentralized finance (DeFi), addressing critical challenges such as scalability, consumer protection, front-running, and stablecoin stability. By bridging the gap between technological advancements and regulatory needs, the research provides innovative solutions to enhance DeFi's accessibility, security, and scalability.

The study investigates fast withdrawal mechanisms in optimistic rollups, enabling users to bypass the traditional seven-day dispute period through tradeable exits. By implementing and analyzing these exits on platforms like Arbitrum, the work evaluates their efficiency, scalability, and risks, offering practical insights into dispute management.

Decentralized order books form another key focus, with a detailed examination of their feasibility, performance, and front-running vulnerabilities. Through the implementation of the Lissy exchange on Ethereum and Layer 2 solutions, the research demonstrates significant improvements in gas efficiency and scalability while proposing novel strategies to mitigate transaction manipulation.

The dissertation also provides a systematized framework for understanding stablecoins, categorizing their stability mechanisms and highlighting vulnerabilities. This analysis lays the groundwork for assessing their role in mitigating volatility and enhancing financial inclusion.

Overall, this work contributes to DeFi's maturation by addressing technical and regulatory challenges, ensuring user-centric design while promoting financial innovation. The findings aim to align DeFi with consumer protection frameworks, paving the way for its broader adoption as a reliable alternative to traditional financial systems.

Acknowledgments

I would like to start by thanking my PhD advisor, Professor Jeremy Clark. Jeremy has been a constant source of patience, wisdom, and encouragement throughout my academic journey. When I first arrived in Canada and began my graduate studies, he guided me through what felt like uncharted territory. His support over the years has helped me grow not just as a researcher but as a person, and I will always be grateful for his mentorship and belief in me.

I am also deeply appreciative of my committee members. Their thoughtful feedbacks and insights helped shape my research and allowed me to see my work from new perspectives. Each of them played an important role in helping me reach this milestone.

To the Offchain Labs family, thank you for being such a wonderful group of people to work with. Your energy, passion, and generosity made even the hardest challenges worthwhile. A special thanks to our CTO, Harry Kalodner, whose guidance and patience have been invaluable over the past two years. I've learned so much from him, and his mentorship has been a highlight of my professional growth.

I also want to thank my co-authors for their collaboration and dedication. The experience of working together—sharing ideas, refining concepts, and tackling challenges—has been an incredibly rewarding part of this journey, and I'm so thankful for their contributions.

Finally, I want to express my deepest gratitude for the support of my partner, Kaveh. Over the past decade, we've shared an incredible journey, starting as two international students far from home, navigating a new world together. Through all the uncertainty and challenges, his unwavering belief in me, his quiet strength, and his constant encouragement have been my anchor. His support has been a foundation for everything I've achieved—from offering a listening ear during hard times to celebrating every milestone along the way. This dissertation is as much a reflection of his love and partnership as it is of my efforts. I am endlessly grateful for the life we've built and the strength we've found together.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Traditional finance (TradFi) has been thoroughly studied over centuries, with few stones left unturned. It is backed by well-established regulations, market structures, and intermediaries. There are courses, certifications, and vast bodies of research dedicated to its systems, ensuring that the complexities of TradFi are well understood. Whether it's buying stocks, obtaining loans, or transferring funds, these systems offer trust, security, and reliability. However, they are often slow, costly, and dependent on multiple intermediaries, which can be inefficient for individual users. While TradFi works well for large institutions, it can be inaccessible or cumbersome for smaller participants, especially in global contexts where financial inclusion remains limited.

In contrast, Decentralized Finance (DeFi) offers a new, permissionless financial system built on blockchain technology, removing intermediaries and automating transactions through smart contracts. DeFi promises open access and global reach, but also introduces new challenges—particularly for end users. Complex interfaces, high gas fees, front-running attacks, limited recourse after losses, and volatile stablecoin behavior can all be significant barriers for users trying to interact with DeFi safely and effectively.

This thesis takes a user-centric approach to addressing these emerging challenges in DeFi. By focusing on real pain points—such as slow withdrawal times, adversarial market behaviors, and stability risks—this work proposes technical solutions that aim to make DeFi systems safer, more intuitive, and more aligned with users' needs. The overarching goal is to enhance consumer protection in decentralized environments that currently lack strong regulatory safeguards or user-support infrastructure.

Although DeFi has gained significant attention recently, with dedicated workshops like the Financial Cryptography Workshop on Decentralized Finance (DeFi), government funding from FRQNT (Fonds de recherche du Québec), and various educational programs, it was not as popular when I started my research. What began as a personal curiosity grew into a commitment to explore the emerging potential of DeFi. After hearing about it, I attended several bootcamps to deepen my knowledge. I presented the early results of my research at FC's 1st Workshop on DeFi in 2021. Since then, I have been actively involved in the program committee for the workshop, staying engaged with the latest developments. In traditional finance, buying a Google share involves choosing from a variety of markets based on liquidity, trading hours, and regulatory frameworks. In DeFi, however, the mechanisms are entirely different, driven by decentralized protocols with blockchain-specific constraints, such as gas fees and scalability. These fundamental differences need deeper exploration. Table 1.1 provides a comparison of key aspects between DeFi and TradFi.

Aspect		**TradFi**	**DeFi**
Low Barriers to Entry	**Users**	TradFi has low technical barriers for users but higher regulatory ones (*e.g.,* KYC/AML compliance).	DeFi has low access barriers but high technical ones (*e.g.,* managing wallets, gas fees, and front-running risks).
	Banks/Fintechs	Banks and fintechs have lower technical barriers in TradFi because they operate within an established financial ecosystem. This infrastructure already supports basic financial services like transactions and account management, making it easier to set up and maintain operations. .	DeFi has high technical barriers for banks and fintechs, as they need to understand the technologies, smart contract deployment, and protocol integration.
Disintermediation		TradFi uses multiple intermediaries, such as brokers and custodians, who offer specialization and expertise in managing specific financial processes. While they bring valuable knowledge to the system, they also increase delays and costs.	DeFi removes intermediaries, allowing direct peer-to-peer transactions but shifting the responsibility for expertise, such as private key management, to users themselves.
Performance		TradFi is slow due to intermediaries (*e.g.,* stock trade settlement takes days), but its infrastructure can handle high transaction volumes efficiently.	DeFi is fast but its speed comes from removing intermediaries, not from faster technology—blockchains can be slower than traditional systems, *e.g.,* traditional systems can execute trades in nanoseconds. L2 solutions improve scalability but add complexity, requiring further research.
Security Paradigm		TradFi relies on long-standing systems managed by intermediaries (banks, brokers), where access is restricted, making it difficult for an outsider to directly attack the system. However, these systems are slower to adapt and less transparent compared to blockchain networks.	DeFi operates through smart contracts, which function like vending machines in the open—anyone can interact with them, including performing attacks, as there are no gatekeepers. While this increases risks, blockchain transparency ensures that all interactions are logged and traceable.
Consumer Protection		TradFi operates under strict regulations, particularly in the U.S., where the SEC and other agencies ensure market integrity, investor protection, and stability. While these rules prevent fraud, they also add layers of compliance, slowing down operations. TradFi requires adherence to protocols like KYC and AML to maintain transparency and accountability.	DeFi operates largely outside regulatory frameworks, making enforcement difficult. As DeFi grows, regulators are seeking ways to impose rules without undermining decentralization, but enforcement remains a challenge. In 2023, the SEC sued major exchanges like Binance for facilitating unregistered securities trading[1], highlighting the growing conflict between DeFi's principles and traditional legal frameworks.

Table 1.1: Comparison of DeFi and TradFi across key aspects

At the beginning of my PhD, I interned with Quebec's Autorité des Marchés Financiers (AMF), where I was exposed to the regulatory challenges posed by DeFi. While my background as an engineer provided technical insight into decentralized

[1]`https://www.sec.gov/news/press-release/2023-101`

finance, AMF's expertise lay in consumer protection. This revealed a gap between the technology and regulatory frameworks, which this research aims to address. The thesis focuses on bridging the divide between DeFi's rapid technological developments and the need for consumer protection within this emerging financial landscape.

One key area of focus is front-running, a challenge for consumer protection in decentralized exchanges. Front-running allows certain actors to take advantage of users' trades, leading to potential losses for regular participants. The research explores methods to mitigate these risks, emphasizing the importance of safeguarding users in decentralized environments.

Stablecoins are another area of concern. Early on, Tether (USDT) was widely used, but it faced scrutiny due to concerns about its reserves not being fully backed 1:1 by the dollar, and its connection to Bitfinex, which faced legal action from the SEC. At the same time, decentralized alternatives like DAI were presented as more secure, but during a major market crash, users faced liquidity issues, highlighting the risks in decentralized stablecoin systems.

The research also explores decentralized order books and fast exit mechanisms, which were initially challenging to implement on Ethereum's Layer 1 due to high costs. With the emergence of Layer 2 solutions like Arbitrum, these issues became more manageable, offering a more scalable approach. This thesis looks at how Layer 2 can improve efficiency while ensuring consumer protection in decentralized trading systems.

By addressing these key areas—front-running, stablecoin risks, and decentralized exchanges—this thesis aims to provide insights into how DeFi can align with regulatory frameworks to protect users, filling the gap between technical innovation and consumer protection.

1.2 Contributions and Outline

This dissertation is organized into four main chapters, each addressing critical aspects of decentralized finance (DeFi) and their implications for the financial sector.

Chapter 2 offers a background and introduction to the essential concepts necessary for comprehending this dissertation. While each subsequent chapter includes a background section specific to its topic, this Chapter lays the foundational knowledge required for understanding blockchains and DeFi. It covers the key technologies and developments that underpin these fields, providing readers with the context needed to explore the more specialized subjects discussed later in the dissertation.

Chapter 3 focuses on optimistic rollups, which are widely used as an opt-in scalability layer for blockchains like Ethereum. In these systems, Ethereum operates as Layer 1 (L1), while the rollup functions as Layer 2 (L2), offering lower fees and reduced latency. A key issue with optimistic rollups is the 7-day dispute period (withdrawal window), during which transfers of tokens, ETH, and general messages from L2 to L1 are not finalized on L1. This chapter explores methods to bypass this dispute period for ETH withdrawals (exits) from L2, even when direct validation of L2 is not possible. By forking Arbitrum Nitro, we enable tradeable exits on L1 before they are finalized and examine how combining these tradeable exits with prediction markets can provide insurance for non-finalized withdrawals. This allows anyone on L1 to safely accept withdrawn tokens during the dispute period. The chapter also presents an open market mechanism that sets fees for fast withdrawals, allowing users to opt in at any time.

Chapter 4 addresses the longstanding concerns of financial regulators about fully decentralized exchanges (DEX) that operate entirely on-chain without clear regulatory oversight. The success of automated market makers (AMMs) like Uniswap has made these concerns more pressing. AMMs function as lightweight dealer-based trading systems, but they deviate significantly from traditional financial systems, inherently requiring fees and being vulnerable to front-running attacks. This chapter explores three core research questions: (1) Are conventional order book-based, secure, fully decentralized exchanges feasible on public blockchains like Ethereum? (2) What is their performance profile? (3) How do Layer 2 solutions, such as Arbitrum, impact performance? To answer these questions, we implement, benchmark, and experiment with an Ethereum-based call market exchange called Lissy. While we find that current Ethereum limitations prevent high trade throughput (a few hundred executions per block), we demonstrate significant scalability improvements on Arbitrum, achieving a 99.88% reduction in gas costs.

Chapter 5 examines the issue of blockchain front-running, where entities exploit privileged access to upcoming transactions or trades for financial gain. While front-running has been a concern in traditional financial markets since the 1970s, blockchain technology has introduced new variations of this problem due to its decentralized and transparent nature. This chapter introduces a new classification system for blockchain-specific front-running attacks, proposes solutions, and evaluates the vulnerability of different decentralized order books to these attacks.

Ultimately, Chapter 6 provides a comprehensive survey of the stability mechanisms employed by stablecoins to reduce the price volatility seen in early cryptocurrencies like Bitcoin and Ether. Rather than focusing on specific stablecoin projects, which are subject to rapid change, this chapter distills the foundational stability mechanisms common across a broad range of stablecoins. The analysis explores how these core mechanisms adjust exchange rates, the assumptions they rely on, and the risks they do not mitigate, offering a clear and enduring framework for understanding stablecoin stability.

Chapter 2

Background

To understand the topics covered in this dissertation, we first need to look at the basics of blockchains and decentralized finance (DeFi). This chapter explains the key concepts, technologies, and developments that form the foundation of these fields, preparing us for a deeper dive into their applications and potential impacts.

2.1 Blockchain Technology

Blockchain is a distributed database open to anyone who wants to participate, resilient against errors and malicious actions, and operates without a central authority. When a participant views their local copy of the ledger, they can be confident that (i) everyone has the same records and (ii) each record was validated by a majority of participants before being added. Blockchain technology, introduced by Satoshi Nakamoto in 2008 as the foundation for Bitcoin, has gained significant global importance [83].

One major challenge in decentralized networks is creating a consensus mechanism that functions without centralized trust (see Section 2.1.1). Efforts to decentralize started in the 1980s [85], but it was Nakamoto's 2008 solution that successfully used a network of incentivized nodes to maintain and validate a public ledger [83]. In a decentralized network, no single entity has control, and anyone can join or leave at any time.

A full node in the Bitcoin network maintains a copy of the blockchain and verifies transactions and blocks. Miners, which are nodes that create new blocks, are rewarded with cryptocurrency for their efforts (see Section 2.1.1). Transactions are the basic units of the blockchain, organized into sequential blocks (see Section 2.1.1). The order of transactions within these blocks is crucial (see Chapter 5).

Public blockchains offer potential for many applications due to their decentralization, transparency, and immutability. However, they are also slow, expensive, and have practical limitations. They are expected to replace many intermediary entities today, though there is a gap between what people think blockchain can do and its actual capabilities [98]. Developers and system designers must reconsider assumptions and data flows due to the public nature of blockchains.

Blockchain has potential for various applications: financial use cases (*e.g.,* Bitcoin [83]), asset trading and markets [33, 104], insurance and futures [72, 81], tamper-resistant record storage, and online voting [76, 12]. Bitcoin introduced the decentralized ledger concept, which has since expanded to many other applications.

Bitcoin's limitations led to the development of other blockchain technologies like Ethereum, introduced by Vitalik Buterin, which includes smart contracts for programmable transactions [28](see Section 2.2). However, Ethereum faces scalability issues, leading to the exploration of layer 2 protocols [34, 90, 60] to improve transaction speeds and reduce costs.

2.1.1 Consensus Mechanism

A consensus mechanism is a fundamental component of blockchain systems. It enables a decentralized network of participants to agree on a single, consistent view of the system's state, even in the presence of faulty or malicious actors. The study of consensus originates in distributed systems theory, where consensus protocols are typically evaluated based on two key properties:

- **Safety:** Nothing bad ever happens. In the context of consensus, this means that all honest nodes agree on the same value and no two honest nodes decide differently (*i.e.,* the system never forks).
- **Liveness:** Something good eventually happens. This ensures that all honest nodes will eventually decide on a value, even in the face of network delays or transient faults.

In blockchain protocols, safety ensures that the state of the ledger is consistent across all nodes, while liveness guarantees that new transactions will continue to be processed and included in the ledger. Achieving both properties simultaneously in an open and adversarial environment is particularly challenging and forms the basis of blockchain consensus research.

Blockchain networks implement different consensus mechanisms to meet these goals. The most prominent families of consensus protocols in this space are:

- **Proof of Work (PoW):** A Nakamoto-style consensus protocol where miners compete to solve computational puzzles to propose blocks. PoW probabilistically ensures consensus by relying on the majority of hash power being honest.
- **Proof of Stake (PoS):** A family of protocols that selects block proposers based on the amount of cryptocurrency they lock up as stake. PoS aims to provide the same security guarantees as PoW, but with significantly reduced energy consumption.

We discuss both of these consensus mechanisms in detail below.

Proof of Work (PoW)

Proof of Work (PoW) is a blockchain consensus mechanism that validates transactions and secures the network. First used by Bitcoin, PoW involves miners competing to solve complex puzzles. The first miner to solve the puzzle adds a new block of transactions to the blockchain and receives cryptocurrency as a reward. This computational effort helps prevent malicious activities like double-spending. Following is a brief overview of the process (see Figure 2.1).

- **Transaction broadcast**: Transactions are broadcast to the network by users.

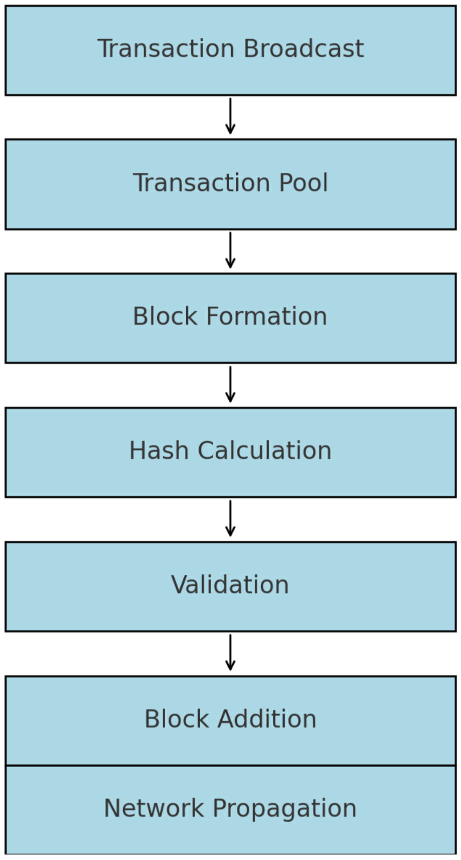


Figure 2.1: Illustration of the proof of work process

- **Transaction pool**: Transactions are collected in a pool (mempool) waiting to be included in a block.

- **Block formation**: Miners gather transactions from the pool and form a block.

- **Hash calculation**: Miners calculate the hash of the block header by varying the nonce.

- **Validation**: The miner checks if the hash meets the target difficulty.

- **Block adddition**: If a valid hash is found, the block is added to the blockchain, and the miner is rewarded.

- **Network propagation**: The new block is propagated to all nodes in the network, which validate and add it to their copy of the blockchain.

To understand PoW, we dive into a few cryptographic primitives and foundational components in the following Sections.

Hash Functions. A hash function is a fundamental component of PoW. A cryptographic hash function $\mathcal{H}(m)$ creates fixed size length outputs called *hash values* for any arbitrary size inputs (*pre-image*)[101]. These functions are used to verify whether a candidate pre-image is equal to the real pre-image value. A perfect cryptographic hash function is non-invertible, meaning that it is infeasible to generate a pre-image from its hash value, this property is referred to as the *pre-image resistance.* Another property of an ideal hash function is called *collision resistance*, that is, it should be infeasible to find two values x and y in such a way that $\mathcal{H}(x) = \mathcal{H}(y)$, and $x \neq y$. In PoW systems, these properties ensure that even a minor alteration in the input results in a vastly different output, making it computationally impractical to forecast the hash value.

Digital Signature. Digital signatures are an essential component of public key cryptography, providing a mechanism for verifying the authenticity and integrity of digital messages. Public key cryptography is an encryption scheme where each user has a pair of keys: a public key (Pk), which is widely known, and a private key (Sk), which is kept secret [38]. Any user can encrypt a plaintext message (M) using the recipient's public key (Pk), and only the corresponding private key can decrypt the resulting ciphertext (C). This process is defined by the encryption and decryption algorithms A_e and A_d, respectively, as shown in Equations 2.1 and 2.2.

$$C = A_{e,Pk}(M) \tag{2.1}$$

$$M = A_{d,Sk}(C) \tag{2.2}$$

Similarly, a digital signature σ acts like a physical signature, proving the authenticity of a message [94]. When Alice signs a message m using her private key (Sk), anyone can verify the signature using her public key (Pk) to confirm that m indeed comes from Alice and has not been altered during transmission. Digital signatures are integral to public key cryptography, with the signing process analogous to encrypting the message with the private key. The creation and verification of digital signatures are represented by Equations 2.3 and 2.4.

$$\sigma = \mathsf{Sign}_{sk}(m) \tag{2.3}$$

$$\mathtt{T/F} = \mathsf{Verify}_{pk}(m, \sigma) \tag{2.4}$$

When users broadcast transactions to the blockchain network, each transaction must be signed by the sender using their private key. This digital signature ensures the authenticity and integrity of the transaction, proving that the sender has authorized it and that the transaction has not been tampered with.

Mining. Mining is the process by which transactions are validated and added to a blockchain in PoW systems. It begins with miners collecting pending transactions from a pool, known as the *mempool*, and assembling them into a candidate block.

The mempool, short for *memory pool*, is a temporary storage area for all pending transactions that have been broadcasted to the network but not yet included in a block. Each node in the network maintains its own mempool, where it holds transactions until they are picked up by a miner. The size and contents of a mempool can fluctuate based on network congestion, transaction fees, and node configurations.

Miners prioritize transactions from the mempool based on various factors, most commonly the transaction fees offered. Transactions with higher fees are typically chosen first, as miners are incentivized by the fees they can collect in addition to the block reward.

However, the mempool is also a focal point for front-running attacks, especially in decentralized finance (DeFi) applications(see Chapter 5). Front-running occurs when someone observes pending transactions in the mempool and submits their own transaction with a higher fee, ensuring it gets processed first. This allows the attacker to exploit information about upcoming transactions, such as large trades, for their gain. For example, an attacker might see a large buy order for a cryptocurrency in the mempool and quickly place their own buy order before the original transaction is confirmed, driving up the price and allowing them to sell at a higher price once the original transaction goes through.

Each miner then attempts to solve a complex mathematical puzzle by finding a nonce, a random value, that when combined with the block's data and hashed using a cryptographic hash function like SHA-256, produces a hash that meets a specific difficulty target set by the network. This target ensures that the hash starts with a certain number of leading zeros, making it computationally intensive to find a valid solution. The first miner to solve the puzzle broadcasts the new block to the network. Other nodes verify the solution, ensuring it adheres to the rules and that the transactions are valid. Upon confirmation, the block is added to the blockchain, and the successful miner is rewarded with newly minted cryptocurrency and transaction fees, incentivizing their participation and securing the network.

However, PoW systems are vulnerable to a 51% attack, where a single entity or group of miners gains control of more than 50% of the network's mining power. This majority control allows them to manipulate the blockchain by reversing transactions, leading to double-spending, and preventing new transactions from being confirmed. Such an attack undermines the trust and integrity of the blockchain, highlighting the importance of a decentralized and widely distributed mining network.

Merkle Trees. Merkle trees, or hash trees, are essential to blockchain systems for ensuring data integrity and verification efficiency. In a Merkle tree, individual transaction hashes are paired and hashed together, creating a binary tree structure where each non-leaf node is the hash of its child nodes. This process continues up the tree until a single hash, called the Merkle root, is generated. The Merkle root acts as a cryptographic commitment to all transactions within a block. This structure allows for quick and secure verification of transaction integrity. For instance, verifying a single transaction requires only a small subset of the tree's hashes rather than the entire dataset, which is crucial for blockchain scalability and security. Additionally, Merkle trees ensure that any modification to a single transaction results in a completely different Merkle root, immediately indicating data tampering. This integrity check is vital for the robustness of PoW systems and the overall security of the blockchain.

Proof of Stake (PoS)

Proof of Stake (PoS) is another consensus mechanism used in blockchain networks as an alternative to Proof of Work (PoW). In PoS, validators are chosen to create new blocks and validate transactions based on the number of coins they hold and are willing to *stake* as collateral. This system reduces the need for extensive computational power and energy consumption required by PoW.

In PoS, validators are incentivized to act honestly, as they risk losing their staked coins if they validate fraudulent transactions. The selection process can be randomized or follow specific rules, ensuring a fair distribution of validation opportunities. By relying on economic incentives rather than computational effort, PoS aims to increase the security, scalability, and sustainability of blockchain networks.

Here's a step-by-step explanation of the Proof of Stake (PoS) process, as illustrated in Figure 2.2).

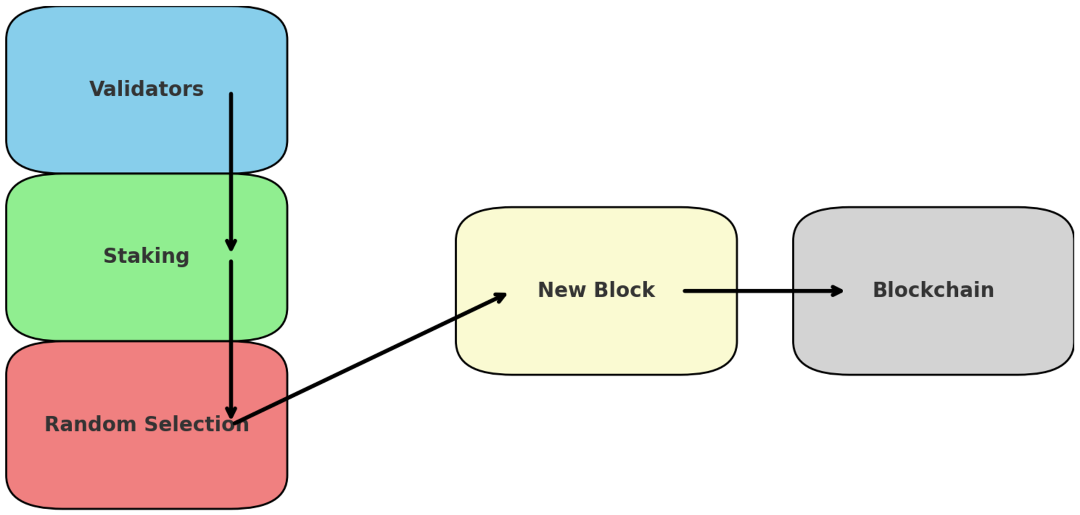


Figure 2.2: Illustration of the proof of stake process

- Validators stake their coins as a security deposit.
- A random selection process picks one of the stakers to validate the next block.
- The selected validator creates a new block and adds it to the blockchain.
- The blockchain grows as each new block is validated and added, maintaining the network's integrity and security.

2.2 Ethereum

The blockchain technology has primitively gained a wide deployment in the area of transactions of digital currencies, such as Bitcoin. However, in 2014, Vitalik Buterin introduced a new blockchain-based application known as Ethereum in his article

Ethereum: A Next-Generation Cryptocurrency and Decentralized Application Platform [28]. As a blockchain-based distributed public network, Ethereum implements a decentralized virtual machine, known as the Ethereum Virtual Machine (EVM), which allows network nodes to execute and deploy programmable smart contracts to the Ethereum blockchain [118]. This new platform enables developers to create and execute blockchain applications called decentralized applications (dapps) in a more efficient way.

Decentralized applications are completely open-source, and their data is stored in a decentralized manner on the blockchain network. Dapps are created by smart contracts, self-executing contracts that are written in a high-level programming language called Solidity, which is similar to C and JavaScript [1]. Digital smart contracts were first described by Nick Szabo in 1993 [108], however, they reached a high level of adoption through blockchain technology.

Ethereum initially employed a Proof of Work (PoW) consensus mechanism, similar to Bitcoin, where miners compete to solve complex mathematical problems to validate transactions and secure the network. However, this process is energy-intensive and has scalability limitations. To address these issues, Ethereum fully transitioned to a Proof of Stake (PoS) consensus mechanism in September 2022 during an event known as the *Merge*. As mentioned in Section 2.1.1, in PoS validators are chosen to create new blocks based on the amount of cryptocurrency they hold and are willing to stake as collateral. This system reduces energy consumption and increases the network's scalability and security by relying on economic incentives rather than computational power. The transition to PoS aims to make Ethereum more sustainable and efficient, enhancing its capability to support a wide range of decentralized applications.

2.2.1 Smart Contracts

As mentioned above, Ethereum's smart contracts are implemented using a high-level programming language called Solidity. Solidity is a statically-typed programming language designed for developing smart contracts that run on the Ethereum Virtual Machine (EVM). Smart contracts written in Solidity are compiled into EVM bytecode, a low-level set of instructions that the EVM can execute.

The process of implementing and executing smart contracts involves several steps:

1. **Writing the Smart Contract:** Developers write the smart contract code in Solidity. This code defines the contract's functions, state variables, and the logic for how the contract should behave.

2. **Compiling to Bytecode:** The Solidity code is compiled into EVM bytecode by the Solidity compiler. This bytecode is a low-level representation of the contract that can be executed by the EVM. The compiler also generates an Application Binary Interface (ABI), which describes how to interact with the contract's functions and events.

3. **Deploying the Contract:** The compiled bytecode and ABI are used to deploy the smart contract to the Ethereum blockchain. Deployment involves creating a transaction that includes the bytecode and sending it to the network. Once

included in a block, the contract is assigned a unique address on the blockchain.

4. **Running the Contract:** Once deployed, the smart contract's functions can be called by sending transactions to its address. When a transaction triggers a function in the contract, the EVM executes the corresponding bytecode. The EVM processes the instructions in the bytecode, interacting with the blockchain's state and updating the contract's storage as necessary.

The execution of EVM bytecode is based on a stack-based architecture, where operations are performed using a last-in, first-out (LIFO) stack. The EVM includes various opcodes for different operations, such as arithmetic, control flow, and interacting with the blockchain's state. Table 2.1 outlines some common EVM opcodes.

The translation from Solidity to EVM bytecode and the execution of this bytecode on the EVM are crucial for the functionality of decentralized applications on Ethereum. This process ensures that smart contracts are executed in a decentralized, secure, and predictable manner.

Opcode	**Description**
`ADD`	Adds two values from the stack.
`SUB`	Subtracts the top value on the stack from the second top value.
`SLOAD`	Loads a value from the contract's storage.
`SSTORE`	Stores a value in the contract's storage.
`CALL`	Calls another contract.
`RETURN`	Returns a value from the contract.

Table 2.1: Common EVM opcodes

2.2.2 Ethereum Accounts

Ethereum has two types of accounts— (i) externally owned accounts and (ii) contract accounts. These accounts are associated with a 40-character hexadecimal format public key known as Ethereum address. Externally owned accounts (*a.k.a. accounts*), own balance and can send transactions to smart contracts. Contract accounts are created every time a smart contract is deployed on the blockchain and hold balance and the contract storage.

2.2.3 Ethereum Transactions

As mentioned, smart contracts are Ethereum accounts that contain a piece of code and can be executed on the Ethereum Virtual Machine (EVM). Once deployed on the Ethereum, these contracts cannot be executed unless they are called up and triggered by mechanisms known as transactions. The caller can be another contracts or normal accounts. As a result of the transaction, the contract code will be executed on the EVM and its state will be updated accordingly.

2.2.4 Ether

Ether (ETH) is a class of cryptocurrency that supplies ”fuel” for the Ethereum network to execute various operations (*e.g.,* executing DApps). In order for Ethereum nodes to process and execute Ethereum transactions, a transaction fee must be paid to compensate for their computing power. These fees are paid in ETH and are calculated based on the computational resources and the time that is required to execute them.

Ether can be obtained by (1) verifying network transactions (*a.k.a.* mining), where ethers are generated as a reward to miners, or (2) receiving from another Ethereum entity.

2.2.5 Gas Model

DApps on Ethereum execute arbitrary code provided by the owner of the DApp. While this code might be written in a high-level programming language like Solidity, it is compiled to a compact representation (called 'bytecode') that is a set of low-level instructions to the environment (Ethereum virtual machine or EVM). Because different functions will have different complexities, the user running the function pays in proportion to the number of instructions, the complexity of the instructions, and the storage requirements. This means that each operation has a fixed price. Naturally, the operations might be priced in ETH, since it is the on-board currency; however, this would cause the price of computation to be as volatile as Ether itself. Instead, Ethereum uses a pseudo-currency called gas.[1] Each instruction has a fixed price in gas. The function caller will pledge to pay a certain amount of ETH (typically quoted in units of **Gwei**, where *wei* is the smallest transactional unit of ETH) per gas, and miners are free to choose to execute that transaction or ignore it. The function caller is charged for exactly the amount the transaction costs and they cap the maximum they are willing to be charged (*gas limit*)—if the cap is too low to complete the execution, the miner keeps the Gwei and reverts the state of the DApp (as if the

[1] http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#what-is-gas

function never ran).

Gas can be expensive, adding a significant cost to executing transactions on the Ethereum network. Additionally, gas prices are not stable, often fluctuating based on network congestion and other factors, which can lead to unpredictable transaction costs. This instability in gas prices underscores the importance of stablecoins (discussed in Chapter 6) as a means to mitigate the volatility in transaction costs.

In Ethereum, transactions are bundled into blocks. A miner can include as many transactions (typically preferring transactions that bid the highest for gas) that can fit under a predetermined block size limit. This competition for inclusion in blocks can further drive up gas prices, particularly during periods of high network activity.

Gas Refunds. In order to reconstruct the current state of Ethereum's EVM, a node must obtain a copy of every variable change since the genesis block (or a more recent 'checkpoint' that is universally agreed to). For this reason, stored variables persist for a long time and, at first glance, it seems pointless to free up variable storage (and unclear what 'free up' even means). Once the current state of the EVM is established by a node, it can forget about every historical variable changes and only concern itself with the variables that have non-zero value (as a byte-string for non-integers) in the current state (uninitialized variables in Ethereum have the value zero by default). Therefore, freeing up variables will reduce the amount of state Ethereum nodes need to maintain going forward.

For this reason, some EVM operations cost a negative amount of gas. That is, the gas is refunded to the sender at the end of the transaction, however (1) the refund is capped at 50% of the total gas cost of the transaction, and (2) the block gas limit applies to the pre-refunded amount (*i.e.,* a transaction receiving a full refund can cost up to 5.5M gas with a 11M limit). Negative gas operations include:

- `SELFDESTRUCT`. This operation destroys the contract that calls it and refunds its balance (if any) to a designated receiver address. `SELFDESTRUCT` operation does not remove the initial byte code of the contract from the chain. It always refunds 24,000 gas. For example, if a contract A stores a single non-zero integer and contract B stores 100 non-zero integers, the `SELFDESTRUCT` refund for both is the same (24,000 gas).

- `SSTORE`. This operation loads a storage slot with a value. Using `SSTORE` to load a zero into a storage slot means the nodes can start ignoring it (recall that all variables, even if uninitialized, have zero by default). Doing this refunds 15,000 gas per slot.

At the time of this writing, Ethereum transaction receipts only account for the `gasUsed`, which is the total amount of gas units spent during a transaction, and users are not able to obtain the value of the EVM's refund counter from inside the EVM [105].

2.3 Blockchain Scalability Challenges and Solutions

As blockchain networks such as Ethereum expand, the need for processing power and storage capacity also rises, resulting in congestion and elevated transaction fees. The intrinsic limitations of PoW, characterized by its high energy consumption and slow transaction speeds, and even the more efficient PoS, are insufficient to fully meet the scalability requirements of a global decentralized network. In general, scalability in blockchain refers to the network's ability to handle a growing number of transactions efficiently. The primary issues include:

- **Transaction Throughput.** Transaction throughput refers to the number of transactions a blockchain can process per second (TPS). Ethereum's throughput is limited by factors such as block size, gas limits, and consensus mechanisms. Each block, generated every 15 seconds, has a fixed size and gas limit, capping the number of transactions it can handle. High transaction volumes lead to delays and increased costs. PoW requires solving complex puzzles, inherently limiting TPS, while PoS, though more efficient, still introduces latency. Currently, Ethereum's TPS is around 15-30, insufficient for high-demand applications [2].

- **Latency.** The time it takes for a transaction to be confirmed can increase significantly with higher network load, affecting the user experience. For example, during periods of high activity, such as popular NFT drops or DeFi trading

[2]Etherscan: `https://etherscan.io/chart/tx`, accessed June. 2024.

spikes, users can experience delays of several minutes or even hours for their transactions to be processed.

- **Cost.** High demand for transaction processing can lead to increased gas fees, making the network expensive to use for everyday transactions. During peak usage times, gas fees can surge to hundreds of dollars per transaction, pricing out smaller users and making the network less accessible.

To address these scalability challenges, various solutions have been proposed and implemented. These include:

Layer 1 (L1) Solutions. Enhancements made directly to the blockchain protocol to increase its capacity and efficiency. One prominent example is Sharding, which aims to partition the network into smaller, more manageable segments called shards [54]. Each shard can process transactions and execute smart contracts independently, significantly increasing the overall network throughput. By distributing the load across multiple shards, the network can handle a higher volume of transactions in parallel, thus mitigating congestion and reducing latency. However, it also introduces complexities in network design, consensus mechanisms, and cross-shard communication. Ensuring security across shards poses risks such as cross-shard attacks and single-shard takeovers, while managing data availability and cross-shard transactions adds further complexity and potential latency

Layer 2 (L2) Solutions. Off-chain mechanisms are developed to reduce the load on the main blockchain, enhancing scalability without modifying the core protocol [52]. These solutions include *state channels*, *side chains*, and *rollups*. State channels allow participants to conduct numerous transactions off-chain, with only the final state being recorded on the blockchain, thus minimizing on-chain interactions and reducing fees. Side chains operate as independent blockchains that run in parallel to the main chain, enabling the transfer of assets and execution of smart contracts while offloading traffic from the primary network. Rollups, specifically zk-rollups and optimistic rollups, bundle multiple transactions off-chain and submit them as a single transaction to the main chain, significantly reducing gas costs and enhancing transaction throughput. Rollups leverage cryptographic proofs or fraud proofs to ensure the integrity of off-chain computations, thereby maintaining the security guarantees of the main blockchain.

These solutions collectively contribute to enhancing the scalability of blockchain networks, addressing the growing demand for higher transaction throughput and lower latency, while maintaining the security and decentralization that are fundamental to blockchain technology.

2.3.1 Optimistic Rollups

Among the most promising Layer 2 solutions are optimistic rollups, which enhance scalability by processing transactions off the main Ethereum chain while maintaining the security and decentralization of the main chain. As mentioned above, optimistic rollups work by bundling multiple transactions into a single batch and processing them off-chain, reducing the load on the Ethereum network. The term ”optimistic” comes from the assumption that transactions are valid by default. Validators submit the batch to the Ethereum main chain with a cryptographic proof, and if any fraudulent activity is detected, a challenge mechanism corrects it.

A function can be computed off-chain and the new state of the DApp, called a *rollup*, is written back to the blockchain, accompanied by either a proof that the function was executed correctly or a dispute resolution process for incorrectly executed functions. There are two main approaches to this: the first is to use cryptographic proof techniques (*e.g.,* SNARKS [17, 50] and variants [16]), known as zk-rollups[3], which are heavy to compute but considered valid once posted to Ethereum. The second approach is to execute the function in a trusted execution environment (TEE; *e.g.,* Intel SGX) and validate the TEE's quote on-chain (*e.g.,* Ekiden [32]).[4]. This TEE-based approach is mired by recent attacks on SGX, but safer TEE technologies like Intel TXT can be substituted. Although the dispute time delays result in slower transaction finality, optimistic rollups substantially increase performance by decreasing gas costs.

Arbitrum [60] and *Optimism* are the two leading implementations of optimistic rollups. Arbitrum employs a multi-round dispute process that minimizes the `L1` gas costs for resolving disputes. In the event of a transaction dispute, the `L1` cost to

[3]zk stands for zero-knowledge, a slight misnomer: succinct arguments of knowledge that only need to be complete and sound, not zero-knowledge, are used [77].

[4]The TEE-based approach is mired by recent attacks on SGX [65, 68, 25, 97], however, these attacks do not necessarily apply to the specifics of how SGX is used here, and safer TEE technologies like Intel TXT (*cf.* [121]) can be substituted.

resolve it is only a small fraction of the cost of executing the transaction itself. In contrast, with Optimism, the cost of dispute resolution is nearly equivalent to the cost of executing the transaction. Next, we briefly explain the core components of *Arbitrum*, as it is essential for understanding several subsequent chapters.

2.3.2 Arbitrum

Arbitrum achieves scalability through a combination of core components and a set of L1 smart contracts that underpin its operation.

Inbox. In a rollup, transactions to be executed on L2 are recorded in a L1 smart contract called the inbox. The inbox serves as the entry point for transactions into the Arbitrum network. Depending on the system, users might submit to the inbox directly, or they might submit to an offchain service, called a sequencer, that will batch together transactions from many users and pay the L1 fees for posting them into the inbox. Transactions recorded in the inbox (as `calldata`) are not executed on Ethereum; instead, they are executed in a separate environment off the Ethereum chain, called L2. This external environment is designed to reduce fees, increase throughput, and decrease latency.

Outbox. Occasionally (*e.g.,* every 30–60 minutes), validators on L2 will produce a checkpoint of the state of all contracts and accounts in the complete L2 according to the latest transactions and will place this asserted state (called an RBlock) in a contract on L1 called the outbox. Note that anyone with a view of L1 can validate that the sequence of transactions recorded in the inbox produces the asserted RBlock in the outbox. This includes Ethereum itself, but asking it to validate this be equivalent to running the transactions on Ethereum. The key breakthrough is that the assertion will be posted with *evidence* that the RBlock is correct so Ethereum does not have to check completely.

Bridge. A final piece of the L2 infrastructure is a bridge, which facilitates seamless communication between the *Arbitrum* and Ethereum main, and can move ETH, tokens, NFTs, and even arbitrary messages between L1 and L2. The bridge does not need to be trusted because every bridge operation is fully determined by the contents of the inbox. Say that Alice transfers ETH to Bob's address on L2. Bob is now entitled to draw down the ETH from L2 to L1 by submitting a withdrawal request using the same process as any other L2 transaction—placing the transaction in the inbox on L1, having it executed on L2, and seeing it finalized in an RBlock on L1.

An RBlock becomes finalized only if no valid challenge is submitted within the 7-day dispute window. This dispute window is a conservative estimate that allows time for valid challenges to be raised. While there's no precise formula for determining the optimal length, the 7-day period accounts for adversarial conditions such as censorship of challengers, suppression attacks, or submission of a fraudulent RBlock. Under reasonable assumptions, suppressing a valid challenge for a full 7 days is considered infeasible, making this a widely accepted safeguard in optimistic rollup designs (see Chapter 3 for a deeper discussion on dispute windows and their impact on user experience).

Chapter 3

Fast and Furious Withdrawals from Optimistic Rollups

This chapter is based on the paper "Fast and Furious Withdrawals from Optimistic Rollups" published in 5th Conference on Advances in Financial Technologies (AFT 2023). This paper was co-authored by Jeremy Clark, Mehdi Salehi, and Daniel Goldman.

3.1 Introduction

Ethereum-compatible blockchain environments, called Layer 2s (or L2s) [53], have demonstrated an ability to reduce transaction fees by 99–99.9% while preserving the strong guarantees of integrity and availability in the underlying Layer 1 (or L1) blockchain. The subject of this chapters concerns one subcategory of L2 technology called an optimistic rollup. The website *L2 Beat* attempts to capitalize all tokens of known value across the top 25 L2 projects. It finds that the top two L2s are both optimistic rollups, *Arbitrum* and *Optimism*, which respectively account for 50% and 30% of all L2 value—$4B USD at the time of writing. [1]

We will describe the working details of optimistic rollups later in this chapter but here are the main takeaways: currently, rollups are faster and cheaper than Ethereum itself. However, each L2 is essentially an isolated environment that cannot instantly and trustlessly interact with accounts and contracts that are running on either L1 or other L2s. An optimistic rollup project will typically provide a smart contract, called a validating bridge [75], that can trustlessly move ETH (and other tokens and even arbitrary messages) between L1 and its own L2. It implements a transfer by locking the ETH in an L1 contract and minting the equivalent ETH on L2 and assigning it to the user's L2 address. More precisely, L2 ETH is a transferrable claim for L1 ETH from the L1 bridge at the request of the current owner of the L2 claim. Later when the user requests a withdrawal, the ETH will be destroyed on L2 and released by the bridge back onto L1 according to whom its new owner is on L2 at the time of the request. This requires the rollup to convince the L1 bridge contract of whom the current owner of withdrawn ETH is on L2. We provide details later but this process takes time: the bridge has to wait for a period of time called the dispute window. The current default is 7 days in *Arbitrum* and *Optimism*, however the filing of new disputes can extend the window. The bottom line is that users have to wait at least

[1] L2 Beat: `https://l2beat.com/scaling/tvl/`, accessed Oct. 2022.

7 days to draw down ETH from an optimistic rollup.

This chapter addresses the following research questions, The numbering of research questions corresponds to the chapter in which they are discussed. Accordingly, this chapter presents RQ3(a), RQ3(b), etc. This convention is followed throughout the dissertation, with each chapter addressing a distinct research question and its subcomponents.

- **RQ3(a):** For a user seeking to withdraw assets from a L2 (*e.g., Arbitrum*) to a L1 (*e.g.,* Ethereum) without waiting for the default dispute period (*i.e.,* 7 days on *Arbitrum*), is there a solution that enables a faster withdrawal while avoiding the need for a trusted third party and allowing them to opt-in after initiating the withdrawal? They can use **tradeable exits**, where users trade their withdrawal claim (ETH_{XX}, *i.e.,* L2 ETH as a claim for L1 ETH) to someone else who can wait for the dispute period to finish. See Section 3.2.2.

- **RQ3(b):** What are the risks associated with tradeable exits (ETH_{XX}) for (i) the withdrawer and for (ii) a buyer? For the withdrawer, there are no risks, as they simply transfer their claim. However, buying ETH_{XX} (a claim for ETH_{L1}, or L1 ETH) introduces risks like settlement risk (the chance the claim will not finalize) and liquidity risk (difficulty selling ETH_{XX}). These factors make ETH_{XX} inherently riskier than direct ETH_{L1} ownership. See Section 3.4.

- **RQ3(c):** How can buyers manage and mitigate the risks associated with purchasing ETH_{XX}? Buyers can mitigate the risks of holding ETH_{XX} by purchasing it at a discount compared to ETH_{L1}. This discount reflects factors like settlement risk, delivery costs, and the opportunity cost of waiting for the dispute period to finalize. See Section 3.4.

- **RQ3(d):** Is the tradeable exit mechanism scalable, and can it be implemented without incurring excessive gas costs or performance penalties? The tradeable exit mechanism is designed to be scalable (computable in constant-time), with gas costs kept relatively low. Directly modifying the outbox to support tradeable exits adds minimal bytecode, and gas costs for functions like `transferSpender()` and exit execution are measured and manageable. See Section 3.3.1.

3.1.1 Related Work and Preliminaries

Arbitrum is first described at *USENIX Security* [60]. Gudgeon *et al.* provide a systemization of knowledge (SoK) of Layer 2 technology (that largely predates rollups) [53]. McCorry *et al.* provide an SoK that covers rollups and validating bridges [75], while Thibault *et al.* provide a survey specifically about rollups [112]. Some papers implement research solutions on Arbitrum for improved performance: decentralized order books [80] and secure multiparty computation [37]. The idea of tradeable exits predates our work but is hard to pinpoint a source (our contribution is implementation and adding hedges). Further academic work on optimistic rollups and bridges is nascent—we anticipate it will become an important research area.

Other related topics are atomic swaps and prediction markets. Too many papers propose atomic swap protocols to list here but see Zamyatin *et al.* for an SoK of the area (and a new theoretical result) [120]. Decentralized prediction markets proposals predate Ethereum and include Clark *et al.* [33] and Truthcoin [109]. Early Ethereum projects *Augur* and *Gnosis* began as prediction markets.

3.2 Proposed Solution

For simplicity, we will describe a fast exit system for withdrawing ETH from L2, however it works for any L1 native fungible token (*e.g.,* ERC20) that is available for exchange on L1. We discuss challenges of fast exits for non-liquid/non-fungible tokens in Section 3.5.4. Consider an amount of 100 ETH. When this amount is in the user's account on L1, we use the notation 100 ETH_{L1}. When it is in the bridge on L1 and in the user's account on L2, we denote it 100 ETH_{L2}. When the ETH has been withdrawn on L2 and the withdrawal has been asserted in the L1 outbox, but the dispute window is still open, we refer to it as 100 ETH_{XX}. Other transitionary states are possible but not needed for our purposes.

3.2.1 Design Landscape

In Table 3.1, we compare our solution to alternatives in industry and the blockchain (academic and grey) literature that could be used for fast withdrawals.

Type	*Example*	No trusted third party	Within an L1 transaction	Within an L2 rollup	No griefing	No free option	Opt-in anytime	Crosschain or L2-to-L2	L1 gasUsed	L2 gasUsed	Other Fees
Normal Exit (baseline)	Arbitrum	●			●	●			200K	80K	—
Centralized	Binance		●	●	●	●		●	400K	21K	Operator
HTLC Swaps	Celer	●	◐	●				●	625K	92K	
Conditional Transfers	StarkEx	●	●	●					⊥	⊥	Operator
Bridge Tokens	Hop	◐	●	●		●		●	1.8M	300K	Operator
Tradeable Exits	This Work	●	∼	●	●	●	●		200K	80K	Discount
Hedged Tradeable Exits	This Work	●	∼	●	●	●	●		265K	80K	$\text{FAIL}_{\textsf{PM}}$

Table 3.1: Comparing alternatives for fast withdrawals from optimistic rollups for liquid and fungible tokens where ● satisfies the property fully, ◐ partially satisfies the property, and no dot means the property is not satisfied. ⊥ was not measured. For our work, ∼ means we propose how to fully achieve the property but do not by default (see caveats in Section 3.5.1).

Properties

We are interested in solutions that do not require a trusted third party. If trust is acceptable, a centralized exchange that has custody of its users funds is a fast and user-friendly solution. We consider anything faster than the 7-day dispute period as "fast" but take measurements of solutions that can settle within a fully confirmed "L1 transaction" (*e.g.,* minutes) and within a unconfirmed L2 RBlock (*e.g.,* hours). This assumes that all counterparties perform instantly upon request. Settlement is from the perspective of the withdrawer, Alice, only and does not necessarily mean other counterparties will complete within the same timeframe. For example, in many solutions, Alice will have her withdrawn ETH quickly at the expense of a counterparty waiting out the dispute period.

Some solutions require one party to act, followed by an action of the counterparty in a follow-up transaction. This creates the risk that the counterparty aborts the protocol before taking their action. Since it is unknown if the counterparty will act or not, these protocols establish a window of time for the counterparty to act and if the window passes without action, the initial party has to begin the protocol again with a new counterparty. The protocols ensure that funds are never at risk of being lost, stolen, or locked up forever, however the protocols admit two smaller issues. The first issue is that a malicious counterparty could accept to participate with no intention of completing the protocol just to "grief" the party taking the action—wasting their time and possibly gas fees for setting up and tearing down the conditions of the trade. The second issue is that a strategic counterparty can accept to participate and then selectively choose to complete or abort, as well as timing exactly when they choose to complete (within the window), based on price movements or other market information. This is called (somewhat cryptically) a "free option;" finance people might recognize it as akin to being given an American call option for free.

A solution is "opt-in anytime" if the user can withdraw normally and then (say upon realizing for the first time that there is a 7 day dispute window) decide to speed up their transaction. While it is not a design goal of this research, many of these solutions are generic cross-chain transactions (including L2-to-L2 swaps). A drawback of our solution is that it is narrowly scoped to L2-to-L1 withdrawals on rollups. Therefore our solution is not intended as a complete replacement of atomic swaps or the other solutions in Table 3.1. It is designed to be best-in-class only for slow rollup withdraws.

Finally we estimate the costs involved for the seller of $\mathrm{ETH}_{\mathsf{L2}}$. For some protocols, the gas cost of the buyer might differ from the seller depending if its actions are symmetric or not—we comment on this but did not find it interesting enough to put in the table. The more interesting aspect is that many alternatives do require a third party to be involved (we generically call them "operators") and they must be compensated for their actions. In some alternatives, the operators might be not be inherently necessary (*e.g.,* an HTLC swap) but are used in practice (*e.g.,* Celer) to ease friction (*e.g.,* users finding other users to swap with): in this case, we are charitable and do not mark the fee. So the fees are for things fundamental to how the alternative works. We expand more within the discussion of each alternative below.

Alternatives

Centralized Consider Alice who has 100 $\mathrm{ETH}_{\mathsf{L2}}$ and wants 100 $\mathrm{ETH}_{\mathsf{L1}}$ for it. A centralized exchange (*e.g., Coinbase*, *Binance*) can open a market for $\mathrm{ETH}_{\mathsf{L2}}/\mathrm{ETH}_{\mathsf{L1}}$. Alternatively, a bridge might rely on an established set of trustees to relay L2 actions to L1. This is called proof of authority; it is distributed but not decentralized (*i.e.,* not an *open* set of participants). The gas costs consists of Alice transferring her $\mathrm{ETH}_{\mathsf{L2}}$ onto the exchange (withdraw to L1 is paid for by the exchange). An exchange will not be profitable if it offers this for free, therefore it captures a operator fee for the service.

Hash Time Locked Contracts (HTLCs) Assume Bob has 100 ETH_{L1} and is willing to swap with Alice. An atomic swap binds together (i) an L2 transaction moving 100 ETH_{L2} from Alice to Bob and (ii) an L1 transaction moving 100 ETH_{L1} from Bob to Alice. Either both execute or both fail. HTLC is a blockchain-friendly atomic swap protocol. Its main drawback is that it also has a time window where Alice (assuming she is the first mover in the protocol) must wait on Bob, who might abort causing Alice's ETH_{L2} to be locked up while waiting (called the griefing problem), or might watch price movements before deciding to act (called free option problem). Bob needs to monitor both chains so he cannot be an autonomous smart contract. HTLCs can work generically between any two chains capable of hash- and time-locking transaction outputs; this includes between two L2s. Recent work by Nadahalli *et al.* [82] proposes a grief-free atomic swap protocol that eliminates both griefing and free option attacks by design, without relying on premiums or trusted third parties. Their construction ensures that neither party can gain by strategically aborting or delaying the swap. While compelling for cross-chain token exchanges, the protocol does not directly address the challenges of rollup exits, specifically, the need to prove L2 state to a L1 bridge with delayed finality. As a result, we focus on alternative mechanisms, such as tradeable exits, which are tailored to the rollup architecture. The transaction (containing a hashlock and timeout) is slightly more complicated than a standard ETH transfer, requiring smart contract logic on both layers. The measurement based on Celer is not a pure HTLC and uses operators as well for liquidity and staking, but we omit these fees from the table because theoretically Alice and Bob could find each other and perform a pure HTLC with no added infrastructure.

Conditional Transfers The intuition behind a conditional transfer (CT) is that L1-to-L2 messaging (or bridging) is fast even if L2-to-L1 messaging is slow. CT exploits this to build an HTLC-esque swap specifically for withdrawing from rollups (while HTLCs are designed generically for cross-chain swaps). Alice beings by registering her intent to trade 100 ETH_{L2} for 100 ETH_{L1} in a special registry contract on L1, and she locks (*e.g.,* for an hour) 100 ETH_{L2} in escrow on L2. If Bob agrees to the swap, Alice provides him (off-chain) with a signed transaction (called the conditional transfer) that transfers the escrowed 100 ETH_{L2} to Bob, conditioned on Alice having receiving 100 ETH_{L1} in the registry contract on L1. After Bob transfers the ETH_{L1} on L1, this fact can be bridged to the L2 escrow contract (with customization of the rollup's inbox) quickly (recall that L1-to-L2 messaging is fast). The L2 escrow contract will flag that the L1 transaction has paid by Bob, and Bob can broadcast his signed (by Alice) L2 transaction to recover 100 ETH_{L2} from escrow (if Bob broadcasts it before the flag is set, it simply reverts).

In terms of existing implementations, we could not adequately isolate the conditional transfer component from the rest of the bridge to measure gas costs (denoted in the table using a $\perp$ symbol) however it should be slight more expensive than an HTLC as the logic of the transaction is more complex.

Also note that Bob must be a validator on L2 to confirm that the state of the escrow and conditional transfer on L2 will result in him being paid—this is where the speedup really comes from, if he waits for L1 to finalize this, then the transfer happens after the dispute period and it is no different than a normal exit. Consequently, Bob cannot be an autonomous L1 smart contract unable to validate L2 state until it is finalized on L1 (which is the design goal of our alternative: hedged tradeable exits).

Bridge Token A bridge token is not a novel technical innovation but it is a practical market design for supplying bridges with liquidity. Bridges between L1 and L2 can technically be implemented by anyone. It is natural for the inbox/outbox provider to provide a bridge but it is not strictly necessary.

Assume a third party creates a contract on L1 that accepts ETH_{L1} and releases a transferable claim for ETH_{L1}; it creates the same contract on L2. Assume enough of these claims come into circulation that a liquid market for them emerges on both layers. To move ETH_{L2} to ETH_{L1}, Alice starts by trading her ETH_{L2} for a claim to the same amount on L2. She then asks the L2 contract to transfer the claim which it does by burning them and firing an event. An authorized party, called a bonder, notices the event on L2, goes to the L1 contract and mints the same number of claims on L1 for ETH_{L1} and transfers them to Alice's address. Technically the L1 contract is insolvent as more claims exist than actual ETH_{L1} in the contract, but the L2 contract is oversolvent by the same amount. The contracts can be rebalanced (1) through movements in the opposite direction; (2) through a bulk withdrawal after the normal 7-day dispute period; or (3) by incentivize bonders to purposefully rebalance the contracts by burning claims on L1 and minting on L2. To prevent the bonder from maliciously minting tokens on L1 that were not burned on L2, it must post a fidelity bond of equal or greater value. (Alternatively, the bonder can be a trusted party which makes it the same in analysis as a centralized exchange). After the 7-day dispute period, the L1 contract can verify the bonder's actions are consistent with the burns on L2 and release its fidelity bond.

Note that when you collapse this functionality, it is equivalent to the bonder buying ETH_{XX} from Alice for ETH_{L1} and receiving their ETH_{L1} back 7 days later. The extra infrastructure is necessary because today native bridges do not support transferring ETH_{XX}. As in atomic swaps, the bonder can fail to act (griefing) which is worst in this case if Alice cannot 'unburn' her tokens, but there is no free option because Bob is a relay and not a recipient of the tokens. The gas fee measurement is based on Hop and standard token transfers on L1 and L2. The main cost of bridge tokens is paying the bonder (called an operator in the table) who are providing a for-profit service.

3.2.2 Tradeable Exits

Alice wants to withdraw 100 ETH_{L2}. Unlike the other solutions, Bob takes the risk that the exit never finalized and therefore will offer less than 100 ETH_{L1} (say 99.95 ETH_{L1}) for it (this is denoted "discount" in Table 1). Assume Bob has 99.95 ETH_{L1} that will not use until after the dispute window. Bob also runs an L2 validator so he is assured that if Alice withdraws, it is valid and will eventually finalize. With a tradeable exit, the outbox allows Alice to change the recipient of her withdraw from herself to Bob. Thus Alice swaps her pending exit of 100 ETH_{L1} (which we call 100 ETH_{XX}) for Bob's 99.95 ETH_{L1} on L1 (note we discuss the actual difference in price in Section 3.4). Since ETH_{L1} and ETH_{XX} are both on L1, Alice can place an ask price for her ETH_{XX} and the first trader willing to swap can do so atomicly, with no ability to grief or capitalize on a free option. After 7 days, Bob can ask the bridge to transfer the ETH_{L1} to his address, and the bridge checks the outbox to validate that Bob's address is the current owner of the exit.

In our forked bridge, Alice can transfer any of her exits that are in an RBlock (*i.e.,* an asserted L2 state update registered in the outbox). Technically, Bob can check the validity of the withdrawal as soon as it is in the inbox, and not wait 30-60 minutes for an RBlock. However for implementation reasons, it is easier to track an exit based on its place (*i.e.,* Merkle path) in an RBlock, rather than its place in the inbox. When we say a withdrawal is 'fast,' we mean 30-60 minutes (*i.e.,* one L2 rollup).

Tradeable exits can be approximated by a third party L1 contract that does not modify the rollup. In this scenario, a L1 contract would act like a proxy for the exit. Alice would specify that she is exiting 100 ETH_{L2} to the proxy contract address (instead of to her address) and set the proxy contract to forward it to her address (if/when it comes through after 7 days). Before the dispute window closes, she can sign a transaction instructing the proxy contract to forward the exit to Bob instead of to her (while giving Bob signing authority over it). In this way, the exit becomes tradeable. After 7 days, the current owner can ask the proxy to fetch the actual transfer from the bridge and forward it to them. If the exit fails, the bridge will refuse the exit.

Given this option, why modify the bridge/outbox of the rollup? This research work is not intended as a strong endorsement of either approach—the reader can decide between the two approaches. Our intention with this research is to discuss, design, implement, and measure the actual functionality of what is needed. This will be largely the same whether it is placed inside or outside the bridge/outbox. The main advantage of modifying the bridge/outbox is that is backward compatible with existing web3 bridge interfaces and with current user behaviour—if web3 interfaces or users do a slow withdraw, our solution can "bail them out" after the fact. Placing the functionality inside the bridge/outbox is more challenging in some regards (*e.g.,* existing code is complex to understand) but also easier in other regards (*e.g.,* our code has direct access to state variables). An outside contract might require minor changes to the bridge anyways, such as creating public interfaces to state variables or other data (*e.g.,* as one example, we later discuss how a prediction market must be able to query the outbox to know if an RBlock is pending, finalized, or failed, which is not a current feature). By contrast, the main advantage of an outside contract is modularity and reducing complexity (and thus risk) within the bridge.

Our approach provides a solution that is both trustless and flexible. Tradeable exits allow users to pass their withdrawal claim to another party after initiating the withdrawal, without needing to anticipate the need for a fast exit in advance. This mechanism eliminates the need for a third party, as users can opt-in to a fast exit at any time.This directly answers **RQ3(a)**, demonstrating a faster, trustless, and opt-in withdrawal mechanism.

3.2.3 Hedged Tradeable Exits

One remaining issue with tradeable exits is how specialized Bob is: he must have liquidity in $\mathrm{ETH}_{\mathsf{L1}}$ (or worst, every token being withdrawn from L2), be online and active, know how to price derivatives, and be a L2 validator. While we can expect blockchain participants with each specialization, it is a lot to assume of a single entity. The goal of this subsection is to split Bob into two distinct participants: Carol and David. Our goal is to allow Carol who does not (or functionally cannot) know anything about L2's current state to safely accept a tradeable exit as if it were equivalent to finalized $\mathrm{ETH}_{\mathsf{L1}}$ (or L1 tokens). Carol could be a L1 contract that accepts the withdrawn tokens for a service or enables exchange. In order to make Carol agnostic of L2, we need David to be aware of L2: David is a L2 validator who understands the risks of an RBlock failing and is willing to bet against it happening. Therefore David needs to also have some liquidity to bet with however it could be $\mathrm{ETH}_{\mathsf{L1}}$ or a stablecoin, while Alice and Carol can interact with all sorts of tokens that David need not heard of or even ones David would not want to hold himself.

Recall that Alice wants $\mathrm{ETH}_{\mathsf{L1}}$ quickly in order to do something on L1 with it; Carol can be that destination contract. The primary risk for Carol accepting $\mathrm{ETH}_{\mathsf{XX}}$ as if it were $\mathrm{ETH}_{\mathsf{L1}}$ is that the RBlock containing the $\mathrm{ETH}_{\mathsf{XX}}$ withdrawal fails and the exit is worthless. If Alice can obtain insurance for the ETH_{XX} that can be verified via $\mathsf{L1}$, then Carol's risk is hedged and she could accept ETH_{XX}. The insurance could take different forms but we propose using a prediction market.

Prediction markets A decentralized prediction market is an autonomous (*e.g.,* vending machine-esque) third party contract. Since we are insuring $\mathsf{L1}$ ETH_{XX}, we need to run the market on $\mathsf{L1}$ (despite the fact that it would be cheaper and faster on $\mathsf{L2}$). Consider a simple market structure based on [33]. A user can request that a new market is created for a given RBlock. The market checks the outbox for the RBlock and its current status (which must be pending). Once opened, any user can submit 1 ETH_{L1} (for example, the actual amount would be smaller but harder to read) and receive two 'shares': one that is a bet that the RBlock will finalize, called $\text{FINAL}_{\mathsf{PM}}$, and one that is a bet that the RBlock will fail, called $\text{FAIL}_{\mathsf{PM}}$. These shares can be traded on any platform. At any time while the prediction market is open, any user can redeem 1 $\text{FINAL}_{\mathsf{PM}}$ and 1 $\text{FAIL}_{\mathsf{PM}}$ for 1 ETH_{L1}. Once the dispute period is over, any user can request that the market close. The market checks the rollup's outbox for the status of the RBlock—since both contacts are on $\mathsf{L1}$, this can be done directly without oracles or governance. If the RBlock finalizes, it offers 1 ETH_{L1} for any 1 $\text{FINAL}_{\mathsf{PM}}$ (and conversely if it fails). The market always has enough ETH_{L1} to fully settle all outstanding shares.

It is argued in the prediction market literature [33] that (i) the price of one share matches the probability (according to the collective wisdom of the market) that its winning condition will occur, and (ii) the price of 1 $\text{FINAL}_{\textsf{PM}}$ and 1 $\text{FAIL}_{\textsf{PM}}$ will sum up to 1 $\text{ETH}_{\textsf{L1}}$. For example, if $\text{FAIL}_{\textsf{PM}}$ trades for 0.001 $\text{ETH}_{\textsf{L1}}$, then (i) the market believes the RBlock will fail with probability of 0.1% and (ii) $\text{FINAL}_{\textsf{PM}}$ will trade for 0.999 $\text{ETH}_{\textsf{L1}}$. These arguments do not assume market friction: if the gas cost for redeeming shares is D (for delivery cost), both share prices will incorporate D (see Section 3.4). Lastly, prediction markets are flexible and traders can enter and exit positions at any time—profiting when they correctly identify over- or under-valued forecasts. This is in contrast to an insurance-esque arrangement where the insurer is committed to hold their position until completion of the arrangement.

Hedging exits. Given a prediction market, Alice can hedge 100 $\text{ETH}_{\textsf{XX}}$ by obtaining 100 $\text{FAIL}_{\textsf{PM}}$ as insurance. Any autonomous L1 contract (Carol) should be willing to accept a portfolio of 100 $\text{ETH}_{\textsf{XX}}$ and 100 $\text{FAIL}_{\textsf{PM}}$ as a guaranteed delivery of 100 $\text{ETH}_{\textsf{L1}}$ after the dispute period, even if Carol cannot validate the state of L2.

Perhaps surprisingly, this result collapses when withdrawing $\text{ETH}_{\textsf{L2}}$—consider Path 1 through the protocol. Alice withdraws 100 $\text{ETH}_{\textsf{L2}}$ from L2 and obtains 100 $\text{ETH}_{\textsf{XX}}$. Bob creates 100 $\text{FAIL}_{\textsf{PM}}$ and 100 $\text{FINAL}_{\textsf{PM}}$ for a cost of 100 $\text{ETH}_{\textsf{L1}}$. Alice buys 100 $\text{FAIL}_{\textsf{PM}}$ from Bob for a small fee. Alice gives Carol 100 $\text{ETH}_{\textsf{XX}}$ and 100 $\text{FAIL}_{\textsf{PM}}$ and is credited as if she deposited 100 $\text{ETH}_{\textsf{L1}}$. In seven days, Bob gets 100 $\text{ETH}_{\textsf{L1}}$ for his 100 $\text{FINAL}_{\textsf{PM}}$ and Carol gets 100 $\text{ETH}_{\textsf{L1}}$ for her 100 $\text{ETH}_{\textsf{XX}}$. If the RBlock fails, Bob has 0 $\text{ETH}_{\textsf{L1}}$ and Carol has 100 $\text{ETH}_{\textsf{L1}}$ from the 100 $\text{FAIL}_{\textsf{PM}}$. In both cases, Alice has a balance of 100 $\text{ETH}_{\textsf{L1}}$ with Carol.

In path 2, Alice withdraws 100 ETH_{L2} from L2 and obtains 100 ETH_{XX}. Alice sells 100 ETH_{XX} to Bob for 100 ETH_{L1}. Alice gives Carol 100 ETH_{L1} and is credited with a balance of 100 ETH_{L1}. In 7 days, Bob gets 100 ETH_{L1} for his 100 ETH_{XX} and Carol has 100 ETH_{L1}. If the RBlock fails, Bob has 0 ETH_{L1}, Carol has 100 ETH_{L1}, and Alice has a balance of 100 ETH_{L1} with Carol.

Modulo differing gas costs and market transaction fees, paths 1 and 2 are equivalent. Path 2 does not use a prediction market at all, it only uses basic tradeable exits. Given this, do prediction markets add nothing to tradeable exits? We argue prediction markets still have value for a few reasons. (1) Speculators will also participate in the prediction market which gives Alice a chance for a fast exit even without Bob (an L2 validator). (2) If Alice withdraws a token other than ETH, the prediction market should still be set up to payout in ETH (otherwise you end up with 50 separate prediction markets for the 50 different kinds of tokens in any given RBlock). In this case, Alice can obtain $\text{FAIL}_{\mathsf{PM}}$ when Bob has no liquidity or interest in the token she is withdrawing (however Carol needs to incorporate an exchange rate risk when accepting an exit in one token and the insurance in ETH). (3) The PM can also help with NFTs and other non-liquid tokens (see Section 3.5.4).

Three of the most common types of traders are utility traders, speculators, and dealers [55]. With a prediction market, Alice is a utility trader and Bob is a dealer. However, there might exist speculators who want to participate in the market because they have forecasts about rollup technology, a given RBlock, the potential for software errors in the rollup or in the validator software, *etc.* Executives of rollup companies could receive bonuses in $\text{FINAL}_{\mathsf{PM}}$. Quick validators might profit from noticing an invalid RBlock with $\text{FAIL}_{\mathsf{PM}}$ or they might be betting on an implementation bug or weeklong censorship of the network. Speculators add liquidity to the prediction market which reduces transactional fees for Alice. However, speculation also brings externalities to the rollup system where the side-bets on an RBlock could exceed the staking requirements for posting an RBlock, breaking the crypo-economic arguments for the rollup. In reality, these externalities can never be prevented in any decentralized incentive-based system [45]. This concern has been formalized in a theoretical result by Ford and Böhme [46], who show that in permissionless systems, prediction markets can create incentives for participants to induce failure deliberately. In our context, a well-capitalized actor could profit more from holding $\text{FAIL}_{\mathsf{PM}}$ shares than they lose from sabotaging a rollup assertion, especially if the market size exceeds the stake or bond protecting the rollup. This introduces a fundamental tension: allowing people to bet on failure also incentivizes causing it. While we expect such behavior to be rare due to economic and reputational costs, any design based on failure-hedging must consider this risk and apply safeguards such as bounding market exposure, adding slashing penalties, or limiting who can interact with the market contract.

3.3 Implementation and Performance Measurements

We run *Arbitrum Nitro* testnet locally and use Hardhat [47] for our experiments. We obtain our performance metrics using TypeScripts scripts.

3.3.1 Tradeable Exits

Trading the exit directly through the bridge/outbox. We fork the *Arbitrum Nitro* outbox to add native support for tradeable exits. The modified outbox is open source, written in 294 lines (SLOC) of Solidity, and a bytecode of 6,212 bytes (increased by 1,197 bytes). The solidity code and Hardhat scripts are available in a GitHub repository.[2] Our modifications include:

- Adding the `transferSpender()` function which allows the exit owner to transfer the exit to any `L1` address even though the dispute period is not passed.
- Adding the `isTransferred()` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is a boolean.
- Adding the `transferredToAddress` mapping which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the current owner of the exit.
- Modifying the `executeTransactionImpl()` function. Once the dispute period is passed and the withdrawal transaction is confirmed, anyone can call the `executeTransaction()` function from the outbox (which internally calls the `executeTransactionImpl()`) and release the funds to the account that was specified by the user 7 days earlier in the `L2` withdrawal request. With our modifications, this function is now enabled to release the requested funds to the

[2]GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

current owner of the exit.

To execute the `transferSpender()` function; Alice (who has initiated a withdrawal for 100 $\text{ETH}_{\texttt{L2}}$) has to provide variables related to her exit (*e.g.,* exit number), which she can query using the Arbitrum SDK[3], as well as the `L1` address she wants to transfer her exit to. The `transferSpender()` function then checks (1) if the exit is already spent, (2) it is already transferred, and (3) the exit is actually a leaf in any unconfirmed RBlock. If the exit has been transferred, the `msg.sender` is cross-checked against the current owner of the exit (recall exit owners are tracked in the `transferredToAddress` mapping added to the outbox). Once these tests are successfully passed, the `transferSpender()` function updates the exit owner by changing the address in the `transferredToAddress` mapping. This costs 85,945 units of `L1` gas. Note that the first transfer always costs more as the user has to pay for initializing the `transferredToAddress` mapping. `transferSpender()` costs 48,810 and 48,798 units of `L1` gas for the second and third transfer respectively. The `gasUed` for executing the new `executeTransactionImpl()` function is 91,418 units of `L1` gas.

Trading the exit through an `L1` market. We also implement and deploy an `L1` market that allows users to trade their exits on `L1` even though the dispute window is not passed (see Section 3.5.3 for why *Uniswap* is not appropriate). In addition, we add a new function to the *Arbitrum Nitro* outbox, the `checkExitOwner()`, which returns the current owner of the exit. Figure 3.1 illustrates an overview of participant

[3]A typescript library for client-side interactions with Arbitrum.

interactions and related gas costs. To start trading, Alice needs to lock her exit up in the market by calling the `transferSpender()` function from the outbox. Next, she can open a market on this exit by calling the `openMarket()` from the market contract and providing the ask price. The market checks if Alice has locked her exit (by calling the `checkExitOwner()` from the outbox) and only in that case a listing is created on this exit. The market would be open until a trade occurs or Alice calls the `closeMarket()` on her exit. Bob, who is willing to buy Alice's exit, calls the payable `submitBid()` function from the market contract. If the `msg.value` is equal or greater than Alice's ask price, the trade occurs; (1) the market calls the `transferSpender()` from the outbox providing Bob's address. Note that market can only do that since it is the current owner of the exit being traded, and (2) the `msg.value` is transferred to Alice.

The market and modified outbox are open source and written in 125 and 294 lines (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the Hardhat scripts are available in a GitHub repository.[4] Once deployed, the bytecode of the market and outbox is 5,772 and 6,264 bytes respectively.

The implementation details demonstrate that the tradeable exit mechanism is scalable and efficient, with low gas costs for transferring exits and executing transactions. The minimal impact on bytecode size and efficient use of gas make it feasible to handle high transaction volumes without incurring excessive costs or performance penalties. This confirms **RQ3(d)**, demonstrating that tradeable exits are scalable

[4]GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

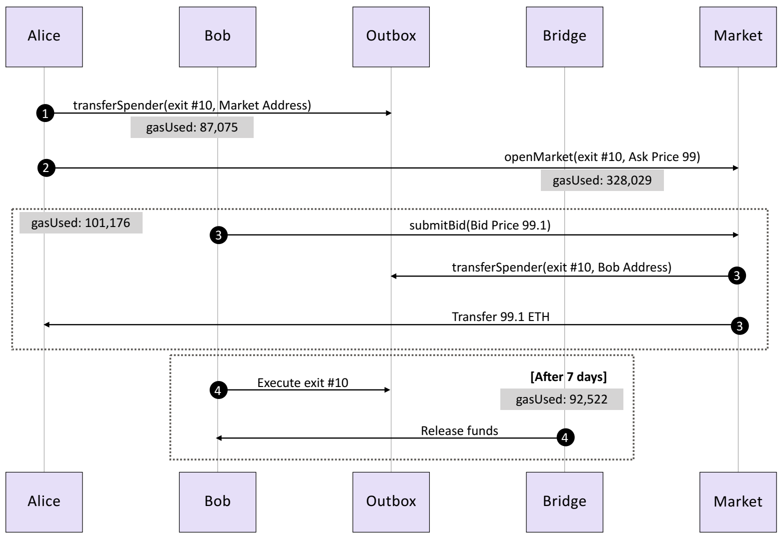


Figure 3.1: Overview of trading the exit through a `L1` market

and can be implemented with low gas and computational overhead.

3.3.2 Prediction Market

As described in Section 3.2.3, a prediction market can be used to hedge the exit. We do not implement this as one can use an existing decentralized prediction market (*e.g., Augur* or *Gnosis*). However, we further modify *Arbitrum Nitro* to make it friendly to a prediction market that wants to learn the status of an RBlock (pending, confirmed). More specifically, we modify the *Arbitrum Nitro* outbox and RollupCore smart contracts, modifications include:

- Adding the `assertionAtState` mapping to the outbox which stores key-value pairs efficiently. The key of the mapping is the exit number and the value is the user-defined data type `state` that restricts the variable to have only one of the `pending and confirmed` predefined values.

- Adding the `markAsPending` function to the outbox which accepts an RBlock and marks it as pending in the `assertionAtState` mapping.

- Adding the `markAsConfirmed` function to the outbox which accepts an RBlock and marks it as confirmed in the `assertionAtState` mapping.

- Modifying the `createNewNode()` function in the RollupCore contract. To propose an RBlock, the validator acts through the RollupCore contract by calling a `createNewNode()` function. We modify this function to call the `markAsPending()` from the outbox which marks the RBlock as pending.

- Modifying the `confirmNode()` function in the RollupCore contract. Once an RBlock is confirmed, the validator acts through the RollupCore contract via `confirmNode` to move the now confirmed RBlock to the outbox. We modify this function to call the `markAsConfirmed()` from the outbox which marks the RBlock as confirmed.

The modified outbox and RollupCore are open source and written in 297 and 560 lines (SLOC) of Solidity respectively. The solidity code for these contracts in addition to the Hardhat scripts are available in a GitHub repository.[5] Once deployed, the bytecode of the outbox and RollupCore is 6,434 and 3,099 bytes respectively.

[5]GitHub:Nitro, Fast-Withdrawals: `https://github.com/MadibaGroup/nitro/tree/fast-withdrawals`

3.4 Pricing

Pricing ETH_{XX}. Consider how much you would pay for 100 ETH_{XX} (finalized in 7 days = 168 hours) in ETH_{L1} today. Since ETH_{XX} is less flexible than ETH_{L1}, it is likely that you do not prefer it to ETH_{L1}, so our intuition is that it should be priced less (*e.g.,* 100 ETH_{XX} = 99 ETH_{L1}). However, our solution works for any pricing and we can even contrive corner cases where ETH_{XX} might be worth more than ETH_{L1} by understanding the factors underlying the price.

In traditional finance [58], forward contracts (and futures, which are standardized, exchange traded forwards) are very similar to ETH_{XX} in that they price today the delivery of an asset or commodity at some future date. One key difference is that with a forward contract, the price is decided today but the actual money is exchanged for the asset at delivery time. When ETH_{XX} is sold for ETH_{L1}, both price determination and the exchange happen today, while the delivery of ETH_{L1} for ETH_{XX} happens in the future. The consequence is that we can adapt pricing equations for forwards/futures, however, the signs (positive/negative) of certain terms need to be inverted.

We review the factors [58] that determine the price of a forward contract (F_0) and translate what they mean for ETH_{XX}:

- *Spot price of ETH_{L1} (S_0):* the price today of what will be delivered in the future. ETH_{XX} is the future delivery of ETH_{L1}, which is by definition worth 100 ETH_{L1} today.

- *Settlement time (Δt):* the time until the exit can be traded for ETH_{L1}. In *Arbi-*

trum, the time depends on whether disputes happen. We simplify by assuming Δt is always 7 days (168 hours) from the assertion time. A known fact about forwards is that F_0 and S_0 converge as Δt approaches 0.

- *Storage cost (U):* most relevant for commodities, receiving delivery of a commodity at a future date relieves the buyer of paying to store it in the short-term. Securing ETH_{XX} and securing ETH_{L1} is identical in normal circumstances, so not having to take possession of ETH_{L1} for Δt time does not reduce costs for a ETH_{XX} holder.

- *Delivery cost (D):* the cost of delivery of the asset, which in our case will encompass gas costs. Exchanging ETH_{L1} for ETH_{XX} requires a transaction fee and also creates a future transaction fee to process the exit (comparable in cost to purchasing a token from an automated market maker). An ETH_{L1} seller should be compensated for these costs in the price of ETH_{XX}.

- *Exchange rate risk:* a relevant factor when the asset being delivered is different than the asset paying for the forward. In our case, we are determining the price in ETH_{L1} for future delivery of ETH_{L1}, thus, there is no exchange risk at this level of the transaction. However, the price of gas (in the term D) is subject to ETH/gas exchange rates. For simplicity, we assume this is built into D.

- *Interest / Yield ($-r + y$):* both ETH_{L1} and ETH_{XX} have the potential to earn interest or yield (compounding over Δt), while for other tokens, there might be an opportunity to earn new tokens simply by holding the token. Let r be the (risk-free) interest (yield) rate for ETH_{L1} that cannot be earned by ETH_{XX}, while y is the opposite: yield earned from ETH_{XX} and not ETH_{L1}. Initially $y > 1$ and $r = 0$, however, with ETH_{XX} becoming mainstream, it is possible $r = y$ (especially hedged ETH_{XX}).

- *Settlement risk (R):* the probability that ETH_{L1} will fail to be delivered for ETH_{XX} discounts the price of ETH_{XX}. We will deal with this separately.

Put together, the price of ETH_{XX} (F_0) is:

$$F_0 = (S_0 + U - D) \cdot e^{(-r+y)\cdot\Delta t} \cdot R$$

This value, F_0, is an expected value—the product of the value and the probability that the RBlock fails to finalize. However, the trader is informed because they have run verification software and checked that the RBlock validates.

$$R = (1 - \mathbf{Pr}[\textsf{rblock}\text{ fails to finalize}|\textsf{rblock}\text{ passes software verification}])$$

Working Example. We start with R. For an RBlock to be up for consideration, it must be submitted to the outbox as a potential solution and for it to fail, a dispute must be filed with an alternative RBlock that the L1 outbox deems to be correct. In our case, the buyer of ETH_{XX} actually runs a L2 validator and thus performs software validation on the RBlock, and will not accept it if the software does not validate it.

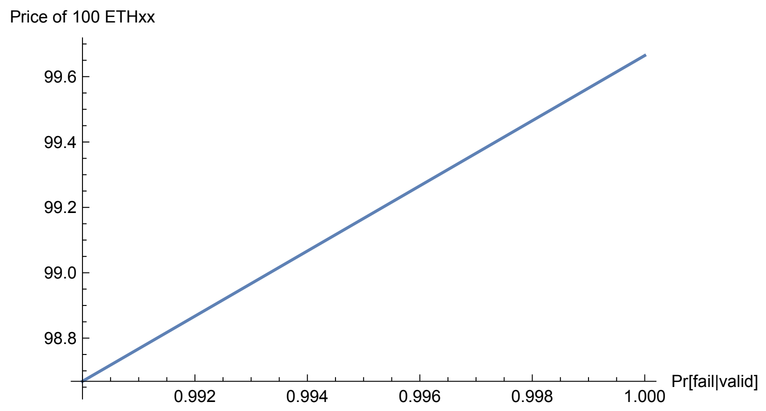


Figure 3.2: Price of 100 $\mathrm{ETH}_{\mathsf{XX}}$ (in ETH) as the probability an RBlock actually finalizes (given the validator checks it with software validation) varies from 99% to 100%, which is denoted by R. Note that 99% is an extraordinarily low probability for this event (considering an RBlock has never failed at the time of writing). The take-away is that the price is not very sensitive to how precisely we estimate R.

For an RBlock to fail given the software validation, it software must have an error that causes a discrepancy between it and the L1 outbox. Furthermore, at least one other validator would need to have different, correct software, and this validator would need to be paying attention to this specific RBlock and independently check it. This should be a rare event and assume $R = (1 - 10^{-15})$ for this example. Figure 3.2 shows a range of R values.

Next, consider the resulting price of F_0. Alice starts with 100 $\mathrm{ETH}_{\mathsf{XX}}$ and Bob purchases it from her. Bob can hold $\mathrm{ETH}_{\mathsf{XX}}$ with no cost ($U = 0$). Alice pays the transaction fee for the deposit, however the cost for the contract for exiting $\mathrm{ETH}_{\mathsf{XX}}$ into $\mathrm{ETH}_{\mathsf{L1}}$ after the dispute period is expected to be $D = 0.008$ ETH (D). Assume a safe-ish annual percent yield (APY) on ETH deposits is 0.2%. Assume $\mathrm{ETH}_{\mathsf{XX}}$ expires in 6 days (0.0164 years). $\mathrm{ETH}_{\mathsf{XX}}$ earns no yield ($y = 0$). Plugging this into the equation, $F_0 = 99.665$ ETH.

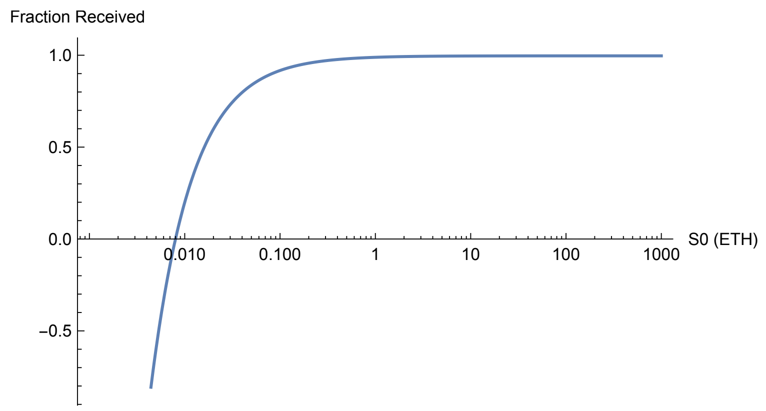


Figure 3.3: This chart shows the percentage of ETH recovered (F_0/S_0) as the amount withdrawn (S_0) increases (log scale), demonstrating it is only economical for withdrawing larger amounts of ETH_{L2}. At low values, the gas costs of a withdrawal dominate. At very low values, the gas costs exceed the price of ETH_{XX} causing the curve to go negative.

As a second example, consider a smaller amount like 0.05 ETH_{XX} (less than \$100 USD at time of writing). Now the gas costs are more dominating. $F_0 = 0.04186$ ETH_{L1} which is only 83.7%. This demonstrates that fast exits are expensive for withdrawals of amounts in the hundreds of dollars. Figure 3.3 shows a range of withdraw amounts.

Lastly, could ETH_{XX} ever be worth more than ETH_{L1}? The equation says yes: with a sufficiently high U or y. A contrived example would be some time-deferral reason (*e.g.,* tax avoidance) to prefer receiving ETH_{L1} in 7 days instead of today. However, in order to purchase ETH_{XX} at a premium to ETH_{L1}, it would have to be cheaper to trade for it than to simply manufacture it. Someone holding ETH_{L1} and wanting ETH_{XX} could simply move it to L2 and then immediately withdraw it to create ETH_{XX}. The gas cost of this path will be one upper bound on how much ETH_{XX} could exceed ETH_{L1} in value.

In summary, holding tradeable exits (ETH_{XX}) instead of direct Layer 1 ETH (ETH_{L1}) introduces specific risks, including settlement risk, price volatility, liquidity challenges, and dependency on Layer 2 infrastructure—factors that make ETH_{XX} inherently riskier than ETH_{L1}. To manage and mitigate these risks, users can price ETH_{XX} at a discount relative to ETH_{L1} to account for settlement and gas costs, providing a buffer against uncertainties.This analysis addresses **RQ3(b)** by identifying the risks associated with holding ETH_{XX}, and answers **RQ3(c)** by showing how buyers can mitigate those risks through discount pricing and optional hedging strategies.

Pricing $\textbf{FINAL}_{\mathsf{PM}}$ and $\textbf{FAIL}_{\mathsf{PM}}$. It might appear surprising at first, but one of the main results of this research is that the price of 100 ETH_{XX} and the of price 100 $\text{FINAL}_{\mathsf{PM}}$ are essentially the same. Both are instruments that are redeemable at the same future time for the same amount of ETH_{L1} (either 100 if the RBlock finalizes and 0 if the RBlock fails) with the same probability of failure (that the RBlock fails). The carrying costs of both are identical. There may be slight differences in the gas costs of redeeming ETH_{L1} once the dispute period is over. However, the operation (at a computational level) is largely the same process. This is actually a natural result: if 100 $\text{FAIL}_{\mathsf{PM}}$ perfectly hedges (reduces the risk to zero) the failure of 100 ETH_{XX} to finalize, then the compliment to $\text{FAIL}_{\mathsf{PM}}$, $\text{FINAL}_{\mathsf{PM}}$, should be priced the same as ETH_{XX}.

3.5 Discussion

3.5.1 Prediction Market Fidelity

A prediction market that covers a larger event should attract more interest and liquidity. For example, betting on an entire RBlock will have more market interest than betting on Alice's specific exit. On the other hand, if markets are exit-specific, the market can be established immediately after Alice's withdrawal hits the inbox instead of waiting for an RBlock (hence $\sim$ in Table 3.1 to indicate it could be done within one L1 transaction). Another consideration arrises when tokens other than ETH are being withdrawn—if the payout of the market matches the withdrawn token, $\text{FAIL}_{\mathsf{PM}}$ will perfectly hedge the exit. Otherwise the hedge is in the equivalent amount of ETH which could change over 7 days. Our suggestion is to promote the most traders in a single market and avoid fragmentation—so we suggest one market in one payout currency (ETH) for one entire RBlock.

3.5.2 Withdrawal Format

As implemented, transferable exits can only be transferred in their entirety. If Alice wants to withdraw 100 ETH_{L2} and give 50 ETH_{XX} to one person and 50 ETH_{XX} to another, she cannot change this once she has initiated the withdraw (if she anticipates it, she can request two separate withdrawals for the smaller amounts). We could implement divisible exits and for ETH; there are no foreseen challenges since the semantics of ETH_{L1} are specified at the protocol-level of Ethereum. However for custom tokens, the bridge would need to know how divisible (if at all) a token is. In fact, a bridge should ensure that the L2 behavior of the tokens is the same as L1 (or that any inconsistencies are not meaningful). Even if a token implementation is standard, such as ERC20, this only ensures it realizes a certain interface (function names and parameters) and does not mean the functions themselves are implemented as expected (parasitic ERC20 contracts are sometimes used to trick automated trading bots.[6] The end result is that bridges today do not allow arbitrary tokens; they are built with allowlists of tokens that are human-reviewed and added by an authorized developer. In this case, ensuring divisible exits are not more divisible than the underlying token should be feasible, but we have not implemented it.

3.5.3 Markets

At the time of writing, the most common way of exchanging tokens on-chain is with an automated market maker (AMM) (*e.g., Uniswap*). If Alice withdraws ETH_{XX} and Bob is a willing buyer with ETH_{L1}, an AMM is not the best market type for them to arrange a trade. AMMs use liquidity providers (LPs) who provide both token types: Alice has ETH_{XX} but no ETH_{L1} that she is willing to lock up (hence why she is trying to fast exit). Bob has ETH_{L1} but to be an LP, he would also need to have ETH_{XX} from another user. However, this only pushes the problem to how Bob got ETH_{XX} from that user. The first user to sell ETH_{XX} cannot use an AMM without locking up ETH_{L1}, which is equivalent to selling ETH_{XX} to herself for ETH_{L1}. The

[6] "Bad Sandwich: DeFi Trader 'Poisons' Front-Running Miners for $250K Profit." *Coindesk*, Mar 2021.

second challenge of an AMM is the unlikely case that an RBlock fails and ETH_{XX} is worthless—then the LPs have to race to withdraw their collateral before other users extract it with worthless ETH_{XX}. It is better to use a traditional order-based market; however, these are expensive to run on L1 [80]. One could do the matchmaking on L2 and then have the buyer and seller execute on L1, but this reintroduces the griefing attacks we have tried to avoid. For now, we implement a very simple one-sided market where Alice can deposit her ETH_{XX} and an offer price, and Bob can later execute the trade against. If Alice is unsure how to price ETH_{XX}, an auction mechanism could be used instead.

3.5.4 Low Liquidity or Non-Fungible Tokens

For tokens that have low liquidity on L1, or in the extreme case, are unique (*e.g.,* an NFT), fast exits do not seem feasible. All the fast exit methods we examined do not actually withdraw the original tokens faster; they substitute a functionally equivalent token that is already on L1. However, we can still help out with low-liquidity withdrawals. We should consider *why* the user wants a fast exit. If it is to sell the token, they can sell the exit instead of the token to any buyer that is L2-aware and willing to wait 7 days to take actual possession. To sell to an L2-agnostic buyer, the seller can insure the exit with enough $\text{FAIL}_{\mathsf{PM}}$ to cover the purchase price. In this case, the buyer does not get the NFT if the RBlock fails but they get their money back.

3.6 Concluding Remarks

The 7-day dispute period in L2 optimistic rollups on Ethereum limits the speed of withdrawals, creating demand for faster, trustless solutions. This chapter explored the tradeable exit mechanism as a flexible alternative, allowing users to bypass wait times by trading their exits for ETH or bundling them with insurance against rollup block failures.

We addressed essential questions around tradeable exits: enabling withdrawals without a third party and allowing users to opt-in to trading after initiating the withdrawal (**RQ3(a)**); understanding the risks involved in holding tradeable exits, such as settlement and liquidity risks (**RQ3(b)**); strategies for pricing and mitigating these risks (**RQ3(c)**); and confirming that tradeable exits can be implemented at scale with manageable gas costs (**RQ3(d)**).

As L2 technology advances, tradeable exits present a viable, scalable solution to the withdrawal delay challenge, with potential applications across multiple L2s and interoperability enhancements to further extend their utility.

Chapter 4

Lissy: Experimenting with On-chain Order Books

This chapter is based on the paper "Lissy: Experimenting with on-chain order books" published in 6th Workshop on Trusted Smart Contracts (WTSC 2022). This paper was co-authored by Jeremy Clark.

4.1 Introduction

There are three main approaches to arranging a trade [55]. In a *quote-driven* market, a dealer uses its own inventory to offer a price for buying or selling an asset. In a *brokered exchange*, a broker finds a buyer and seller. In an *order-driven* market, offers to buy (*bids*) and sell (*offers/asks*) from many traders are placed as orders in an order book. Order-driven markets can be *continuous*, with buyers/sellers at any time adding orders to the order book (*makers*) or executing against an existing order (*takers*); or they can be *called*, where all traders submit orders within a window of time and orders are matched in a batch (like an auction).

Conventional financial markets (*e.g.,* NYSE, NASDAQ) use both continuous time trading during open hours, and a call market before and during open hours to establish an opening price and a closing price. After early experiments at implementing continuous time trading on Ethereum (*e.g.,* EtherDelta, OasisDEX), it was generally accepted that conventional trading is infeasible on Ethereum for performance reasons. Centralized exchanges continued their predominance, while slowly some exchanges moved partial functionality on-chain (*e.g.,* custody of assets) while executing trades off-chain.

A clever quote-driven alternative, called an automatic market maker (AMM), was developed that only requires data structures and traversals with low gas complexity. This approach has undesirable price dynamics (*e.g.,* market impact of a trade, slippage between the best bid/ask and actual average execution price, *etc.*) which explains why there is no Wall Street equivalent, however, it is efficient on Ethereum and works 'good enough' to attract trading. First generation AMMs provide makers (called liquidity providers) with no ability to act on price information—they are uninformed traders that can only lose (called impermanent loss) on trades but make money on fees. Current generation AMMs (*e.g.,* Uniswap v3) provided informed makers with a limited ability (called concentrated liquidity) to act on proprietary information [92] without breaking Ethereum's performance limitations. Ironically, the logical extension of this is a move back to where it all started—a full-fledged order-driven exchange that allows informed makers the fullest ability to trade strategically.

This chapter addresses the following research questions:

- **RQ4(a):** What type of exchange has the fairest price execution on balance? A call market. See Section 4.2 for more explanation.
- **RQ4(b):** How many orders can be processed on-chain? Upper-bounded by 152 per block. See Section 4.2.3.
- **RQ4(c):** How much efficiency can be squeezed from diligently choosing the best data structures? Somewhat limited; turn 38 trades into 152. See Section 4.2.1.
- **RQ4(d):** Can we stop the exchange's storage footprint on Ethereum from bloating? Yes, but it is so expensive that it is not worth it. See Section 4.2.2.
- **RQ4(e):** Are on-chain order books feasible on layer 2? Yes! Optimistic roll-ups reduce gas costs by 99.88%. See Section 4.3.1.
- **RQ4(f):** Which aspects of Ethereum were encountered that required deeper than surface-level knowledge to navigate? Optimizing gas refunds, Solidity is not truly object-oriented, miner extractable value (MEV) can be leveraged for good, and bridging assets for layer 2. See Section 4.2.1.
- **RQ4(g):** How hard is an on-chain exchange to regulate? The design leaves almost no regulatory hooks beyond miners (and sequencers on layer 2). See Section 4.1.1.

4.1.1 Related Work and Preliminaries

Call markets are studied widely in finance and provide high integrity prices (*e.g.,* closing prices that are highly referenced and used in derivative products) [57, 91, 43]. They can also combat high frequency trading [24, 11]. An older 2014 paper [33] on the 'Princeton prediction market' [22] show that call markets mitigate most blockchain-based front-running attacks present in an on-chain continuous-trading exchange as well as other limitations: block intervals are slow and not continuous, there is no support for accurate time-stamping, transactions can be dropped or reordered by miners, and fast traders can react to submitted orders/cancellations when broadcast to network but not in a block and have their orders appear first. The paper does not include an implementation, was envisioned as running on a custom blockchain (Ethereum was still in development in 2014) and market operations are part of the blockchain logic.

The most similar academic work to this research is the Ethereum-based periodic auction by Galal *et al.* [48] and the continuous-time exchange TEX [64]. As with us, front-running is a main consideration of these works. In a recent SoK on front-running attacks in blockchain [40], three general mitigations are proposed: confidentiality, sequencing, and design. Both of these papers use confidentiality over the content of orders (*cf.* [113, 119, 114, 31, 72]). The main downside is that honest traders cannot submit their orders and leave, they must interact in a second round to reveal their orders. The second mitigation approach is to sequence transactions according to some rule akin to first-in-first-out [63, 67]. These are not available for experimentation on

Ethereum yet (although Chainlink has announced an intention[1]). The third solution is to design the service in a way that front-running attacks are not profitable—this is the approach with Lissy which uses *no cryptography* and is *submit-and-go* for traders. A detailed comparison of front-running is provided in Chapter 5.5. Our research also emphasizes implementation details: Galal *et al.* do not provide a full implementation, and TEX uses both on-chain and off-chain components, and thus does not answer our research question of how feasible an on-chain order book is.

Trade Execution Systems

Centralized Exchanges (CEX). Traditional financial markets (*e.g.,* NYSE and NASDAQ) use order-matching systems to arrange trades. An exchange will list one or more assets (stocks, bonds, derivatives, or more exotic securities) to be traded with each other, given its own order book priced in a currency (*e.g.,* USD). Exchanges for blockchain-based assets (also called crypto assets by enthusiasts) can operate the same way, using a centralized exchange (CEX) design where a firm (*e.g.,* Binance, Bitfinex, *etc.*) operates the platform as a trusted third party in every aspect: custodianship over assets/currency being traded, exchanging assets fairly, offering the best possible price execution. Security breaches and fraud in centralized exchanges (*e.g.,* MtGox [88], QuadrigaCX [102], and many others) have become a common source of lost funds for users, while accusations of unfair trade execution have been leveled but are difficult to prove. Today, CEXes are often regulated as other money ser-

[1]A. Juels. blog.chain.link, 11 Sep 2020.

vice businesses—this provides some ability for the government to conduct financial tracking but does little to provide consumer protection against fraud.

On-chain Order Books. For trades between two blockchain-based assets (*e.g.,* a digital asset priced in a cryptocurrency, stablecoin, or second digital asset), order matching can be performed ”on-chain” by deploying the order-matching system either on a dedicated blockchain or within a decentralized application (DApp). In this model, traders entrust their assets to an autonomously operating DApp with transparent source code, instead of a third-party custodian that could mismanage or lose the funds. The trading rules operate as coded, ensuring reliable clearing and settling, with order submission managed by the blockchain—a reasonably fair and transparent system. Furthermore, anyone can create an on-chain order book for any asset (on the same chain) at any time. Although these advantages are significant, performance remains a key challenge and is the main subject of this chapter. Since it is an open system, there is no clear regulatory hook, beyond the blockchain itself.

On-chain exchanges pose significant regulatory challenges because they operate autonomously without a central intermediary. Unlike centralized exchanges, which can be regulated due to their custodial role and central control, on-chain order books are governed by immutable smart contracts, leaving no centralized entity to oversee or enforce compliance. This decentralized structure complicates traditional regulatory approaches, as mechanisms such as consumer protection or fraud enforcement are difficult to apply. The only potential regulatory leverage lies at the blockchain level itself, which is generally resistant to centralized control or modification. This analysis directly addresses **RQ4(g)** highlighting how the absence of custodial responsibility and central oversight makes decentralized exchanges difficult to regulate.

In this research, we focus on benchmarking an order book on the public blockchain Ethereum. Ethereum is widely used and presents a challenging environment in terms of performance and gas constraints. Exchanges could alternatively be deployed on dedicated blockchains, where the trade execution logic is integrated into the network protocol. Permissioned blockchains (*e.g.,* NASDAQ Linq, tZero) can improve execution time and throughput but may undermine user trust and transparency if left unregulated.

On-chain Dealers. An advantage of on-chain trading is that other smart contracts, not just human users, can initiate trades, enabling broader decentralized finance (DeFi) applications. This has fueled a resurgence in on-chain exchange but through a quote-driven design rather than an order-driven one. Automated market makers (*e.g.,* Uniswap v3) have all the trust advantages of an on-chain order book, plus they are relatively more efficient. The trade-off is that they operate as a dealer—the DApp exchanges assets from its own inventory. This inventory is loaded into the DApp by an investor who will not profit from the trades themselves but hopes their losses (termed 'impermanent losses') are offset over the long-term by trading fees. By contrast, an order book requires no upfront inventory and trading fees are optional. Finally, there is a complicated difference in their price dynamics (*e.g.,* market impact of a trade, slippage between the best bid/ask and actual average execution price, *etc.*)—deserving of an entire research paper to precisely define. We leave it as an assertion that with equal liquidity, order books have more favourable price dynamics for traders.

Hybrid Designs. Before on-chain dealers became prominent in the late 2010s, the most popular design was hybrid order-driven exchanges with some trusted off-chain components and some on-chain functionalities. Such decentralized exchanges (DEXes) were envisioned as operating fully on-chain, but performance limitations drove developers to move key components, such as the order matching system, off-chain to a centralized database. A landscape of DEX designs exist (*e.g.,* EtherDelta, 0x, IDEX, *etc.*): many avoid taking custodianship of assets off-chain, and virtually all (for order-driven markets) operate the order book itself off-chain (a regulatory hook). A non-custodial DEX solves the big issue of a CEX—the operator stealing the funds—however trade execution is still not provably fair, funds can still be indirectly stolen by a malicious exchange executing unauthorized trades, and server downtime is a common frustration for traders.

Table 4.1 presents an overview of different trade execution systems, summarizing their respective advantages and disadvantages. Fully decentralized, on-chain exchanges, which are discussed in detail earlier in this section, require minimal trust, offer instant settlement, and ensure transparent and accurate trading rules. However, they are susceptible to front-running attacks, a significant vulnerability of blockchains

Type	**Description**	**Advantages**	**Disadvantages**
Centralized Exchanges (CEX)	Order-driven exchange acts as a trusted third party (*e.g.,* Binance, Bitfinex)	Conventional Highest performance Low fees Easy to regulate Low price slippage Verbose trading strategies	Fully trusted custodian Slow withdrawals Server downtime Uncertain fair execution
Partially On-chain Exchange	Order-driven exchange acts as a semi-trusted party (*e.g.,* EtherDelta, 0x, IDEX, Loopring)	High performance Low fees Easy to regulate Low price slippage Verbose trading strategies Semi-custodial	Slow withdrawals Server downtime Front-running attacks Uncertain fair execution
On-Chain Dealers	Quote-driven decentralized exchange trades from inventory with public pricing rule (*e.g.,* Uniswap v3)	Non-custodial Instant trading Moderate performance Fair execution	Unconventional Impermanent loss High price slippage Intrinsic fees Front-running attacks Limited trading strategies Hard to regulate
On-chain Order-Driven Exchanges	Order-driven decentralized exchange executes trades between buyers and sellers (*e.g.,* Lissy)	Conventional Non-custodial Low price slippage Fair execution Verbose trading strategies Front-running is mitigable	Very low performance Hard to regulate

Table 4.1: Comparison among different trade execution systems

Time	**Trader**	**Order Type**	**Order Price**	**Volume**
09:10	Mehdi	Ask	10.18	4
09:12	Avni	Bid	12	3
09:15	Kritee	Bid	13	3
09:18	Bob	Bid	12.15	1
09:26	Navjot	Ask	10.15	4
09:30	Alice	Ask	10	1

Table 4.2: Example orders that are submitted to a call market

that necessitates specific mitigation strategies, as thoroughly discussed in Chapter 5.5.

Call Markets

Assume traders submit their orders in Table 4.2 to a call market when it is open. In the following, we explain how these orders are executed:

- The call market first matches Alice's ask order to sell 1 at 10 with Kritee's bid order to buy 3 at 13. Trade occurs at the ask price of 10, and 3 will be given to the miner as price improvement. This trade fills Alice's order and leaves Kritee with a remainder of 2 to buy at 13.

- Next, the call market matches Kritee's remainder of 2 with Navjot's ask order to sell 4 at 10.15. Trade occurs at 10.15 and 2.85 is given to the miner as price improvement. This trade fills 2 units from Navjot's order and completes Kritee's bid, leaving Navjot with a remainder of 2 to sell at 10.15.

- The market now matches the next highest bid, Bob's bid order to buy 1 at 12.15, with Navjot's remaining ask of 2 at 10.15. Trade occurs at 10.15 and 2 is given to the miner as price improvement. This fills Bob's order and leaves Navjot with 1 unit remaining.

- Next, the market matches Avni's bid order to buy 3 at 12 with the remainder of Navjot's ask order to sell 1 at 10.15. Trade occurs at 10.15 and 1.85 is given to the miner as price improvement. This fills 1 unit of Avni's bid and completes Navjot's order, leaving Avni with 2 units to buy at 12.

- Finally, the market matches Avni's remaining 2 units with Mehdi's ask order to sell 4 at 10.18. Trade occurs at 10.18 and 1.82 is given to the miner as price improvement. This completes Avni's order and leaves Mehdi with 2 units unfilled at 10.18.

Operation	**Description**
depositToken()	Deposits ERC20 tokens in Lissy smart contract
depositEther()	Deposits ETH in Lissy smart contract
openMarket()	Opens the market
closeMarket()	Closes the market and processes the orders
submitBid()	Inserts the upcoming bids inside the priority queue
submitAsk()	Inserts the upcoming asks inside the priority queue
claimTokens()	Transfers tokens to the traders
claimEther()	Transfers ETH to the traders

Table 4.3: Primary operations of Lissy smart contract

4.2 Call Market Design

A call market opens for traders to submit bids and asks, which are enqueued until the market closes. Trades are executed by matching the best-priced bid with the best-priced ask, a process that inherently promotes fair price execution. This matching continues until the best bid is lower than the best ask, ensuring that all trades occur at mutually favorable prices. In addition, any difference between a trader's bid or ask price and the actual execution price is allocated to miners as a price improvement. This design inherently promotes fair price execution. See Table 4.2 for a numeric example. If Alice's bid of \$100 is executed against Bob's ask of \$90, Alice pays \$100, Bob receives \$90, and the \$10 difference (called a price improvement) is given to miners, for reasons explained in the front-running evaluation (Chapter 5.5).

This price improvement mechanism also serves as a built-in mitigation against miner extractable value (MEV). In traditional on-chain exchanges, miners (or sequencers) can reorder or front-run transactions to extract value from traders. In Lissy, however, the maximum value that can be extracted through reordering is already explicitly captured and redirected to the block producer via price improvement. This design aligns the block producer's incentives with the protocol's intended behavior, reducing the motivation for adversarial manipulation. A more detailed evaluation of front-running and MEV is provided in Chapter 5.5, but we emphasize here that allocating price improvements to miners transforms potential MEV into a protocol-sanctioned reward for fair execution.

For our experiments and measurements, we implement a call market from scratch. Lissy will open for a specified period of time during which it will accept a capped number of orders (*e.g.,* 100 orders—parameterized so that all orders can be processed), and these orders are added to a priority queue (discussed in Section 4.2.1). Our vision is the market would be open for a very short period of time, close, and then reopen immediately (*e.g.,* every other block). Lissy is open source and written in 336 lines (SLOC) of Solidity plus the priority queue (*e.g.,* we implement 5 variants, each around 300 SLOC). We tested it with the Mocha testing framework using Truffle [107] on Ganache-CLI [106] to obtain our performance metrics. Once deployed, the bytecode of Lissy is 10,812 bytes plus the constructor code (6,400 bytes), which is not stored. The Solidity source code for Lissy and Truffle test files are available in a GitHub repository.[2] We have also deployed Lissy on Ethereum's testnet Rinkeby with flattened (single file) source code of just the Lissy base class and priority queue implementations. It is visible and can be interacted with here: [etherscan.io]. We cross-checked for vulnerabilities with *Slither*[3] and *SmartCheck*[4], and it only fails some 'informational' warnings that are intentional design choices (*e.g.,* a costly loop). All measurements

[2]https://github.com/MadibaGroup/2020-Orderbook
[3]https://github.com/crytic/slither
[4]https://tool.smartdec.net

Operation	**Description**
Enqueue()	Inserts an element into the priority queue
Dequeue()	Removes and returns the highest priority element
isEmpty()	Checks if the priority queue is empty

Table 4.4: Operations for a generic priority queue

assume a block gas limit of 11 741 495 and 1 gas = 56 Gwei.[5] Table 4.3 summarizes Lissy's primary operations. This design directly addresses **RQ4(a)** by demonstrating how a call market enables fair price execution by batching orders and matching them at a uniform clearing price.

4.2.1 Priority Queues

In designing Lissy within Ethereum's gas model, performance is the main bottleneck. For a call market, closing the market and processing all the orders are the most time-consuming steps. Assessing which data structures will perform best is hard (*e.g.,* gas refunds, a relatively cheap mapping data structure, only partial support for object-oriented programming) without actually deploying and evaluating several variants.

We first observe that orders are executed in order: highest to lowest price for bids, and lowest to highest price for asks. This means random access to the data structure holding the orders is unnecessary (we discuss cancelling orders later in Section 4.3.3). We can use a lightweight *priority queue* (PQ) which has only two functions: Enqueue() inserts an element into the priority queue; and Dequeue() removes and returns the highest priority element (Table 4.4). Specifically, we use two PQs—one for bids, where the highest price is the highest priority, and one for asks, where the lowest

[5]EthStats (July 2020): `https://ethstats.net/`

price is the highest priority.

As closing the market is very expensive with any PQ, we rule out sorting the elements while dequeuing and sort during each enqueue. We then implement the following 5 PQ variants, with results indicating that the linked list variants are materially cheaper than heap structures for dequeuing. This selection maximizes the number of processable trades per block, achieving a 4x improvement—turning 38 trades into 152—by carefully choosing the optimal data structures. This directly addresses **RQ4(c)** by showing how selecting efficient data structures allows us to increase the number of executable trades from 38 to 152 within a single block, and also contributes to **RQ4(f)** by illustrating the depth of gas-level and storage-level understanding required to implement on-chain priority queues efficiently.

The design choices made in Lissy revealed specific limitations in Ethereum's gas model and Solidity's partial support for object-oriented programming, showing aspects of Ethereum that require deep technical understanding for performance optimization.

1. **Heap with Dynamic Array.** A heap is a binary tree where data is stored in nodes in a specific order where the root always represents the highest priority item (*i.e.,* highest bid price/lowest ask price). Our heap stores its data in a Solidity-provided dynamically sized array. The theoretical time complexity is logarithmic enqueue and logarithmic dequeue.

2. **Heap with Static Array.** This variant replaces the dynamic array with a So-

lidity storage array where the size is statically allocated. This is asymptotically the same and marginally faster in practice.

3. **Heap with Mapping.** In this variant, we store a key for the order in the heap instead of the entire order. Once a key is dequeued, the order struct is drawn from a Solidity mapping (which stores key-value pairs very efficiently). This is asymptotically the same and faster with variable-sized data.

4. **Linked List.** In this variant, elements are stored in a linked list (enabling us to efficiently insert a new element between two existing elements during enqueue). Solidity is described as object-oriented but the Solidity equivalent of an object is an entire smart contract. Therefore, an object-oriented linked list must either (1) create each node in the list as a struct—but this is not possible as Solidity does not support recursive structs—or (2) make every node in the list its own contract. The latter option seems wasteful and unusual, but it surprisingly ends up being the most gas efficient data structure to dequeue. The theoretical time complexity is linear enqueue and constant dequeue.

5. **Linked List with Mapping.** Finally, we try a variant of a linked list using a Solidity mapping. The value of the mapping is a struct with the incoming order's data and the key of the next (and previous) node in the list. The contract stores the key of the first node (head) and last node (tail) in the list. Asymptotically, it is linear enqueue and constant dequeue.

We implemented, deployed, and tested each PQ. A simple test of enqueuing 50

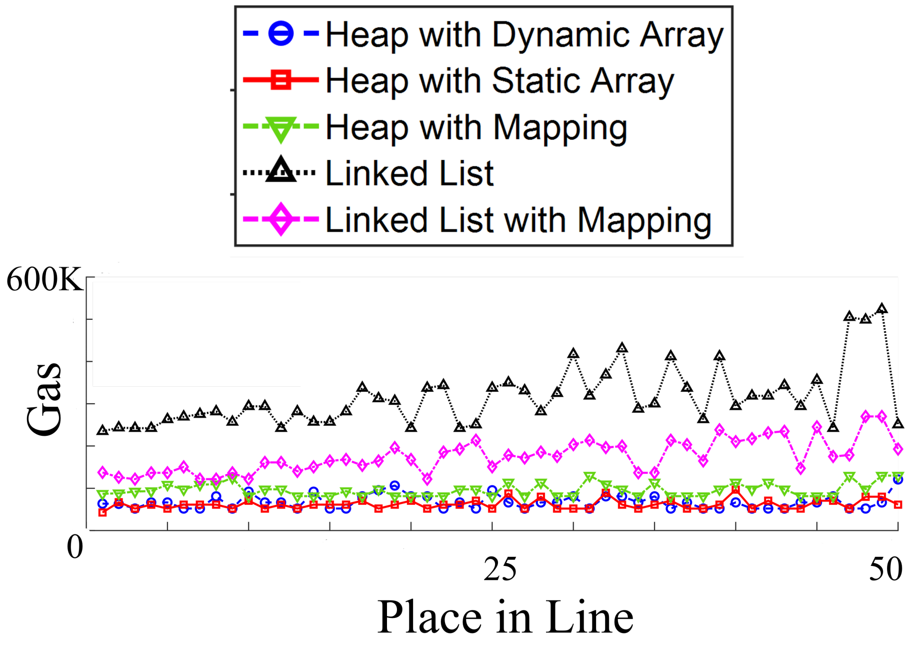


Figure 4.1: Gas costs for enqueuing 50 random integers into five priority queue variants. For the x-axis, a value of 9 indicates it is the 9th integer entered in the priority queue. The y-axis is the cost of enqueuing in gas.

integers chosen at random from a fixed interval is in Figure 4.1 and dequeing them all is in Table 4.5. Dequeuing removes data from the contract's storage resulting in a gas refund. Based on our manual estimates,[6] every variant receives the maximum gas refund possible (*i.e.,* half the total cost of the transaction). In other words, each of them actually consumes twice the `gasUsed` amount in gas before the refund. However, none of them are better or worse based on how much of a refund they generate.

We observe that (1) the linked list variants are materially cheaper than the heap variants at dequeuing; (2) dequeuing in a call market must be done as a batch, whereas enqueuing is paid for one at a time by the trader submitting the order; and

[6]EVM does not expose the refund counter. We determine how many storage slots are being cleared and how many smart contracts destroyed, then we multiply these numbers by 24,000 or 15,000 respectively.

	Gas Used	Refund	Full Refund?
Heap with Dynamic Array	2,518,131	750,000	●
Heap with Static Array	1,385,307	750,000	●
Heap with Mapping	2,781,684	1,500,000	●
Linked List	557,085	1,200,000	●
Linked List with Mapping	731,514	3,765,000	●

Table 4.5: The gas metrics associated with dequeuing 50 integers from five priority queue variants. Full refund amount is shown but the actual refund that is applied is capped.

(3) Ethereum will not permit more than hundreds of orders so asymptotic behaviour is not significant. For these reasons, we suggest using one of the linked list variants. As it can be seen in Figure 4.1, the associated cost for inserting elements into a linked list PQ is significantly greater than the linked list with mapping, as each insertion causes the creation of a new contract. Accordingly, we choose to implement the call market with the linked list with mapping which balances a moderate gas cost for insertion (*i.e.,* order submission) with one for removal (*i.e.,* closing the market and matching the orders). In Section 4.3, we implement Lissy on Layer 2. There, the PQ variant does not change the layer 1 gas costs (as calldata size is the same) and the number of orders can be substantially increased. thus, we reconsider asymptotic and choose a heap (with dynamic array) to lower L2 gas costs across both enqueuing and dequeuing.

	Gas Used	Refund	Full Refund?
Linked List without `SELFDESTRUCT`	721,370	0	◐
Linked List with `SELFDESTRUCT`	557,085	1,200,000	●
Linked List with Mapping and without `DELETE`	334,689	765,000	●
Linked List with Mapping and `DELETE`	731,514	3,765,000	●

Table 4.6: The gas metrics associated with dequeuing 50 integers from four linked list variants. For the refund, (●) indicates the refund was capped at the maximum amount and (◐) means a greater refund would be possible.

4.2.2 Cost/Benefit of Cleaning Up After Yourself

One consequence of a linked list is that a new contract is created for every node in the list. Beyond being expensive for adding new nodes (a cost that will be borne by the trader in a call market), it also leaves a large footprint in the active Ethereum state, especially if we leave the nodes on the blockchain in perpetuity (*i.e.,* we just update the head node of the list and leave the previous head 'dangling'). To address **RQ4(d)**, we explore strategies like using the `SELFDESTRUCT` operation in the `Dequeue()` function to control the exchange's storage footprint on Ethereum, effectively reducing unnecessary storage bloat. As shown in Table 4.6, the refund from doing this outweighs the cost of the extra computation: gas costs are reduced from 721K to 557K. This suggests a general principle: cleaning up after yourself will pay for itself in gas refunds. Unfortunately, this is not universally true, as shown by applying the same principle to the linked list with mapping.

Dequeuing in a linked list with mapping can be implemented in two ways. The simplest approach is to process a node, update the head pointer, and leave the 'removed' node's data behind in the mapping untouched (where it will never be referenced again). Alternatively, we can call `DELETE` on each mapping entry once we finish processing a trade. As it can be seen in the last two rows of Table 4.6, leaving the data on the blockchain is cheaper than cleaning it up.

Addressing **RQ4(f)** also required navigating several technical intricacies of Ethereum. Implementing Lissy within Ethereum's gas model, for instance, demanded a deeper understanding of Solidity's handling of storage and memory, particularly as each node in a linked list required its own contract due to Solidity's limited object-oriented support, which increased storage costs. Additionally, while gas refunds incentivize cleaning up storage variables, the effectiveness of these refunds is highly contextual—often determined by the cap on maximum refunds, which can limit the efficiency of large cleanup operations. Cleaning up EVM state remains a complex and under-explored aspect of Ethereum, with many nuances in managing gas costs and storage efficiently.

For our own work, we strive to be good citizens of Ethereum and clean up to the extent that we can—thus all PQs in Table 4.5 implement some cleanup, demonstrating our approach to managing Ethereum's storage footprint in alignment with these learned principles.

4.2.3 Lissy Performance Measurements

The main research question is how many orders can be processed under the Ethereum block gas limit. The choice of PQ implementation is the main influence on perfor-

	Max Trades (w.c.)	Gas Used for Max Trades	Gas Used for 1000 Trades	Gas Used for Submission (avg)
Heap with Dynamic Array	38	5,372,679	457,326,935	207,932
Heap with Static Array	42	5,247,636	333,656,805	197,710
Heap with Mapping	46	5,285,275	226,499,722	215,040
Linked List	152	5,495,265	35,823,601	735,243
Linked List with Mapping	86	5,433,259	62,774,170	547,466

Table 4.7: Performance of Lissy for each PQ variant. Each consumes just under the block gas limit ($\sim$11M gas) with a full refund of half of its gas.

mance and the results are shown in Table 4.7. These numbers are for the *worst-case*—when every submitted bid and ask is marketable (*i.e.,* will require fulfillment). In practice, once `closeMarket()` hits the first bid or ask that cannot be executed, it can stop processing all remaining orders. Premised on Ethereum becoming more efficient over time, we were interested in how much gas it would cost to execute 1000 pairs of orders, which is given in the third column. The fourth column indicates the cost of submitting a bid or ask — since this cost will vary depending on how many orders are already submitted (recall Figure 4.1), we average the cost of 200 order submissions.

The main takeaway is that call markets on Ethereum are effectively upper-bounded at processing 152 orders per block. Achieving this limit requires nearly the entire block gas limit, which makes it feasible only in specific cases, such as low-liquidity tokens or markets with a limited number of traders (*e.g.,* liquidation auctions). This section answers **RQ4(b)** by empirically measuring how many orders can be processed in a single Ethereum block under current gas limits.

4.3 Lissy on Arbitrum

As discussed in Chapter 2, *Layer 2 (L2)* solutions [52] are a group of scaling technologies proposed to address specific drawbacks of executing transactions on Ethereum, which is considered *Layer 1 (L1)*. Among these proposals, *roll-ups* prioritize reducing gas costs (as opposed to other valid concerns like latency and throughput, which are secondary for Lissy). In a roll-up, every transaction is stored (but not executed) on Ethereum, then executed off-chain, and the independently verifiable result is pushed back to Ethereum, with some evidence of being executed correctly.

We choose to experiment with Lissy on the optimistic rollup Arbitrum.[7] To deploy a DApp on Arbitrum, or to execute a function on an existing Arbitrum DApp, the transaction is sent to an *inbox* on L1. It is not executed on L1, it is only recorded (as calldata) in the inbox. An open network of *validators* watch the inbox for new transactions. Once inbox transactions are finalized in an Ethereum block, validators will execute the transactions and assert the result of the execution to other validators on a sidechain called ArbOS. As the Inbox contract maintains all Arbitrum transactions, anyone can recompute the entire current state of the ArbOS and file a dispute if executions are not correctly reported on ArbOS. Disputes are adjudicated by Ethereum itself and require a small, constant amount of gas, invariant to how expensive the transaction being disputed is. When the dispute challenge period is over, the new state of ArbOS is stored as a checkpoint on Ethereum.

[7]See `https://offchainlabs.com` for more current details than the 2018 *USENIX Security* paper [60].

Figure 4.2 shows how traders interact with Lissy on Arbitrum. First, a trader sends a `depositETH` transaction on Ethereum to the Inbox contract to deposit X amount of ETH to the Arbitrum chain. Once the transaction is confirmed, X amount of ETH will be credited to the trader's address on the Arbitrum chain. Trader can now interact with Lissy and execute its functions by sending the instruction and data required for those executions to either (1) the Arbitrum regular Inbox on Ethereum, or (2) the sequencer. In our example, trader uses the regular Inbox to execute `depositEther()` and the sequencer to execute `submitBid()` from Lissy that lives entirely on Arbitrum chain. Accordingly, trader deposits ETH to Lissy smart contract by sending the instruction and data for executing the `depositEther()` to the Arbitrum Inbox contract that lives on Ethereum. A validator fetches this transaction from the Inbox, executes it, and asserts the result to ArbOS. Next, trader sends the instruction and data for execution of `submitBid()` to the sequencer. The sequencer then inserts this message into the Inbox that it owns. This Inbox contract has the same interface as the regular Inbox contract, however, it is *owned* by the sequencer. A validator sees the transaction in the sequencer Inbox of the bridge, executes it, and asserts the result to ArbOS. Periodically, the entire state of ArbOS is committed back to Ethereum.

Our Lissy variant is not the first roll-up-based order book. Loopring 3.0[8] offers a continuous-time order book. The primary difference is that orders in Loopring 3.0 are submitted off-chain to the operator directly, whereas our variant uses on-chain

[8]`https://loopring.org`

	Layer1 gasUsed	**Layer2 ArbGas**
Lissy on Ethereum	5,372,679	N/A
Lissy on Arbitrum	6,569	508,250

Table 4.8: Gas costs of closing a market on Ethereum and on Arbitrum. ArbGas corresponds to Layer 2 *computation used*.

submission so that the roll-up server does not need to be publicly reachable. Loopring 3.0 can operate near high-frequency trading as order submission is unhampered by Ethereum. However, its roll-up proof does not ensure that the exchange did not reorder transactions, which is particularly problematic in a continuous-time order book. Traders who prioritize trade fairness might opt for a solution like our variant, while traders who want speed would vastly prefer the Loopring architecture which offers near-CEX speed while being non-custodial. Loopring leaves a regulatory hook whereas our variant could be nearly as difficult to regulate as a fully on-chain solution if the roll-up server was kept anonymous: Ethereum and Arbitrum themselves would be the only regulatory hooks.

4.3.1 Lissy Performance Measurements on Arbitrum

Testing Platforms. We implement Lissy using the Arbitrum Rollup chain hosted on the Rinkeby testnet. It is visible and can be interacted with here: [Arbitrum Explorer]. To call functions on Lissy, traders can (1) send transactions directly to the Inbox contract, or (2) use a relay server (called a *Sequencer*) provided by the Arbitrum. The sequencer will group, order, and send all pending transactions together as a single Rinkeby transaction to the Inbox (and pays the gas).

In our Lissy variant on Arbitrum, the validators do all computations (both enqueuing and dequeuing) so we choose to use a heap with dynamic array for our priority queue, which balances the expense of both operations. Heaps are 32% more efficient than linked lists for submitting orders and 29% less efficient for closing. Recall that without a roll-up, such a priority queue can only match 38 pairs at a cost of 5,372,679 gas. Table 4.8 shows that 38 pairs cost only 6,569 in L1 gas (a 99.88% savings). This is the cost of submitting the `closeMarket()` transaction to the Inbox to be recorded, which is 103 bytes of calldata. Most importantly, recording `closeMarket()` in the Inbox will always cost around 6,569 even as the number of trades increases from 38 pairs to thousands or millions of pairs. Of course, as the number of trades increase, the work for the validators on L2 increases, as measured in ArbGas. The price of ArbGas in Gwei is not well established but is anticipated to be relatively cheap. Arbitrum also reduces the costs for traders to submit an order: from 207,932 to 6,917 in L1 gas. We illustrate the interaction between the traders and Lissy on Arbitrum including bridges, inboxes, sequencers and validators in Figure 4.2.

Running Lissy on Arbitrum has one large caveat. If the ERC20 tokens being traded are not issued on ArbOS, which is nearly always the case today, they first need to be *bridged* onto ArbOS, as does the ETH. Traders send ETH or tokens to Arbitrum's bridge contracts which create the equivalent amount at the same address on L2. Withdrawals work the same way in reverse, but are only final on L1 after a dispute challenge period (currently 1 hour).[9]

These results show that Lissy on Arbitrum achieves substantial gas savings and

[9]L1 users might accept assets before they are finalized as they can determine their eventual emergence on L1 is indisputable (*eventual finality*).

scalability, confirming that on-chain order books are indeed feasible on Layer 2. This directly addresses **RQ4(e)**, which asks whether such a design can work under the cost and performance constraints of rollups.

4.3.2 Token Divisibility and Ties

A common trading rule is to fill ties in proportion to their volume (*i.e., pro rata* allocation)[10]. This can fail when tokens are not divisible. Consider the following corner case: 3 equally priced bids of 1 non-divisible token and 1 ask at the same price: (1) the bid could be randomly chosen (*cf.* Libra [73]), or (2) the bid could be prioritized based on time. In Lissy, tokens are assumed to be divisible. If the volume of the current best bid does not match the best ask, the larger order is partially filled and the remaining volume is considered against the next best order. We note the conditions under which pro rata allocation fails (*i.e.,* non-divisible assets, an exact tie on price, and part of the final allocation) are improbable. (1) is the fairest solution with one main drawback: on-chain sources of 'randomness' are generally deterministic and manipulatable by miners [20, 27], while countermeasures can take a few blocks to select [19]. We implement (2) which means front-running attacks are possible in this one improbable case.

[10]If Alice and Bob bid the same price for 100 tokens and 20 tokens respectively, and there are only 60 tokens left in marketable asks, Alice receives 50 and Bob 10.

4.3.3 Order Cancellations

Support for cancellation opens the market to new front-running issues where other traders (or miners) can displace cancellations until after the market closes. However, one benefit of a call market is that beating a cancellation with a new order has no effect, assuming the cancellation is run any time before the market closes. Also, cancellations have a performance impact. Cancelled orders can be removed from the underlying data structure or accumulated in a list that is cross-checked when closing the market. Removing orders requires a more verbose structure than a priority queue (*e.g.,* a self-balancing binary search tree instead of a heap; or methods to traverse a linked list rather than only pulling from the head). Lissy does not support order cancellations. We intend to open and close markets quickly (on the order of blocks), so orders are relatively short-lived.

4.3.4 Who Pays to Close/Reopen the Market?

In the Princeton paper [33], the call market is envisioned as an alt-coin, where orders accumulate within a block and a miner closes the market as part of the logic of producing a new block (*i.e.,* within the same portion of code as computing their coinbase transaction in Bitcoin or `gasUsed` in Ethereum). In Lissy, someone needs to execute `closeMarket()` at the right time and pay for it, which is probably the most significant design challenge for Lissy.

Since price improvements are paid to the miners, the miner is incentivized to run `closeMarket()` if it pays for itself. Efficient algorithms for miners to automatically find 'miner extractable value (MEV)' opportunities [35] is an open research problem. Even if someone else pays to close the market, MEV smooths out some market functionality. Assume several orders are submitted and then `closeMarket()`. A naive miner might order the `closeMarket()` before the submitted orders, effectively killing those orders and hurting its own potential profit. MEV encourages miners to make sure a profitable `closeMarket()` in the mempool executes within its current block (to claim the reward for itself) and that it runs after other orders in the mempool to maximize its profit.

Without MEV, markets should open and close on different blocks. In this alternative, the `closeMarket()` function calls `openMarket()` as a subroutine and sets two modifiers: orders are only accepted in the block immediately after the current block (*i.e.,* the block that executes the `closeMarket()`) and `closeMarket()` cannot be run again until two blocks after the current block.

Another option is to have traders in the next call market pay to incrementally close the current market. For example, each order in the next market needs to pay to execute the next x orders in the current market until the order book is empty. This has two issues: first, amortizing the cost of closing the market amongst the early traders of the new market disincentives trading early in the market; the second issue is if not enough traders submit orders in the new market, the old market never closes (resulting in a backlog of old markets waiting to close).

A closely related option is to levy a carefully computed fee against the traders for every new order they submit. These fees are accumulated by the DApp to use as a bounty. When the time window for the open market elapses, the sender of the first `closeMarket()` function to be confirmed receives the bounty. This is still not perfect: `closeMarket()` cost does not follow a tight linear increase with the number of orders, and gas prices vary over time which could render the bounty insufficient for offsetting the `closeMarket()` cost. If the DApp can pay for its own functions, an interested party can also arrange for a commercial service (*e.g.,* any.sender[11]) to relay the `closeMarket()` function call on Ethereum (an approach called *meta-transactions*). This creates a regulatory hook.

The final option is to rely on an interested third party (such as the token issuer for a given market) to always close the market, or occasionally bailout the market when one of the above mechanisms fails. An external service like Ethereum Alarm Clock[12] (which also creates a regulatory hook) can be used to schedule regular `closeMarket()` calls.

4.3.5 Collateralization Options

in Lissy, both the tokens and ETH that a trader wants to potentially use in the order book are preloaded into the contract. Consider Alice, who holds a token and decides she wants to trade it for ETH. In this model, she must first transfer the tokens to the contract and then submit an ask order. If she does this within the same block, there is a chance that a miner will execute the ask before the transfer and the ask will revert. If she waits for confirmation, this introduces a delay. This delay seems

[11]`https://github.com/PISAresearch/docs.any.sender`

[12]`https://ethereum-alarm-clock-service.readthedocs.io/`

reasonable but we point out a few options it could be addressed:

1. **Use `msg.value`.** For the ETH side of a trade (*i.e.,* for bids), ETH could be sent with the function call to `submitBid()` to remove the need for `depositEther()`. This works for markets that trade ERC20 tokens for ETH, but would not work for ERC20 to ERC20 exchanges.

2. **Merge Deposits with Bids/Asks.** Lissy could have an additional function that atomically runs the functionality of `depositToken()` followed by the functionality of `submitAsk()`. This removes the chance that the deposit and order submission are ordered incorrectly.

3. **Use ERC20 Approval.** Instead of Lissy taking custody of the tokens, the token holder could simply approve Lissy to transfer tokens on her behalf. If Lissy is coded securely, it is unconcerning to allow the approval to stand long-term and the trader never has to lock up their tokens in the DApp. The issue is that there is no guarantee that the tokens are actually available when the market closes (*i.e.,* Alice can approve a DApp to spend 100 tokens even if she only has 5 tokens or no tokens). In this case, Lissy would optimistically try to transfer the tokens and if it fails, move onto the next order. This also gives Alice an indirect way to cancel an order, by removing the tokens backing the order—this could be a feature or it could be considered an abuse.

4. **Use a Fidelity Bond.** Traders could post some number of tokens as a fidelity bond, and be allowed to submit orders up to 100x this value using approve. If

a trade fails because the pledged tokens are not available, the fidelity bond is slashed as punishment. This allows traders to side-step time-consuming transfers to and from Lissy while still incentivizing them to ensure that submitted orders can actually be executed. The trade-off is that Lissy needs to update balances with external calls to the ERC20 contract instead of simply updating its internal ledger.

4.4 Concluding Remarks

Launching a token on Ethereum is only the first step; enabling trading is essential for success. This chapter explored trading methods, with a focus on on-chain order books, which offer immediate, accessible trading without relying on centralized or complex external infrastructure.

We addressed key questions on the feasibility of on-chain order books. Call markets emerged as a fair mechanism for price execution (**RQ4(a)**), though limited to about 152 orders per Ethereum block (**RQ4(b)**). Efficiency gains from optimized data structures (**RQ4(c)**) and strategies to mitigate front-running (**RQ4(d)**) can enhance performance, while using SELFDESTRUCT effectively reduces storage bloat on Ethereum (**RQ4(e)**). Layer 2 solutions like roll-ups make large-scale on-chain trading viable, significantly cutting gas costs (**RQ4(f)**), though the decentralized nature of these exchanges poses regulatory challenges (**RQ4(g)**).

As blockchain scalability advances, on-chain order books may become a preferred trading method, seamlessly bridging token issuance and trading accessibility.

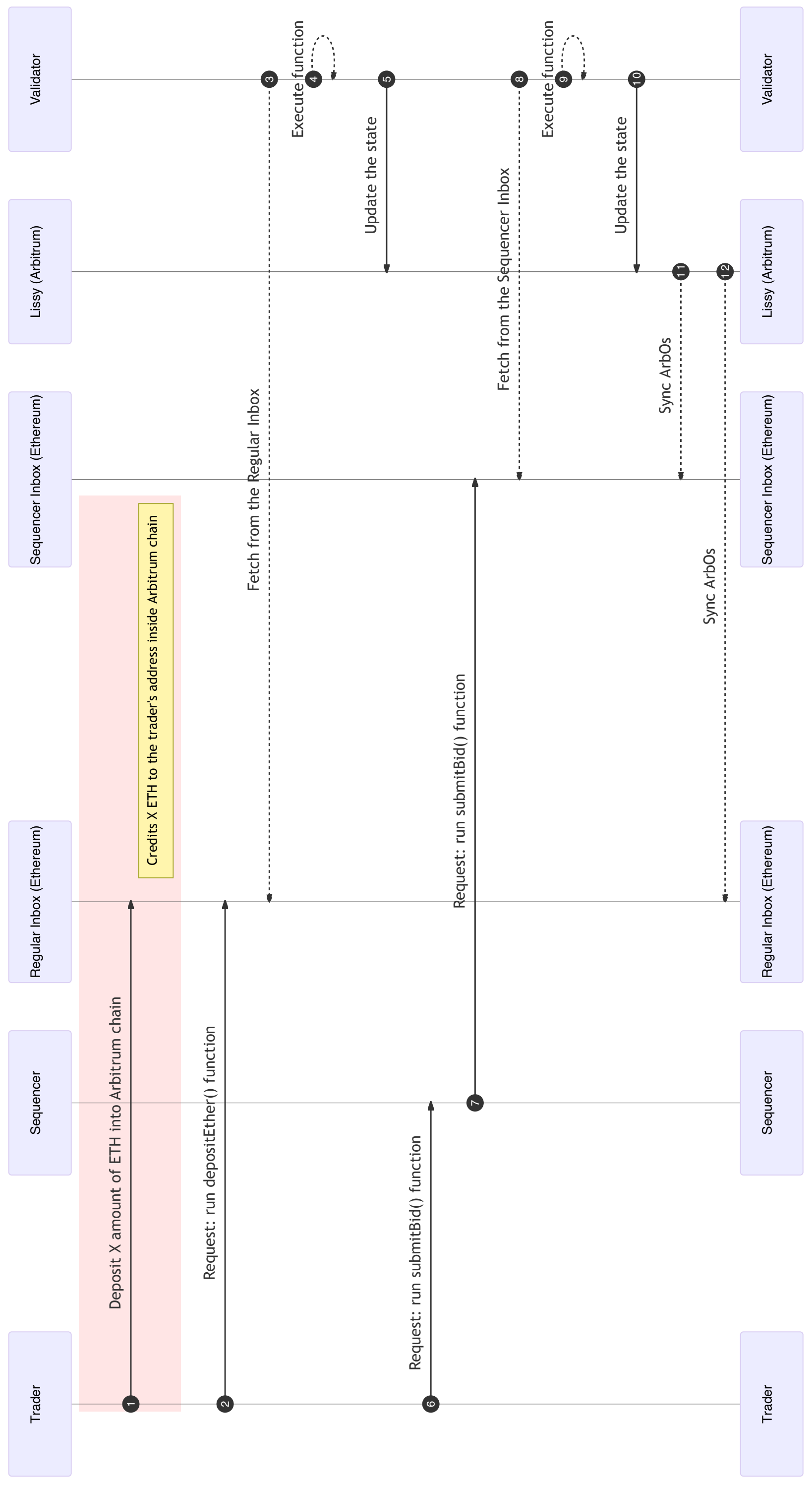

Trader
Sequencer
Regular Inbox (Ethereum)
Sequencer Inbox (Ethereum)
Lissy (Arbitrum)
Validator
1 Deposit X amount of ETH into Arbitrum chain
Credits X ETH to the trader's address inside Arbitrum chain
2 Request: run depositEther() function
3 Fetch from the Regular Inbox
4 Execute function
5 Update the state
6 Request: run submitBid() function
7 Request: run submitBid() function
8 Fetch from the Sequencer Inbox
9 Execute function
10 Update the state
11 Sync ArbOs
12 Sync ArbOs

Figure 4.2: Overview of Lissy on Arbitrum

Chapter 5

Front-running Attacks on On-Chain Order books

This chapter is adapted from the paper "SoK: Transparent Dishonesty: Front-Running Attacks on Blockchain" published at 3rd Workshop on Trusted Smart Contracts In Association with Financial Cryptography (FC) in February 2019. This paper was co-authored with Jeremy Clark and Shayan Eskandari. It merges in relevant sections from the full version of the Lissy paper (Chapter 4).

5.1 Introduction

The key difference between a decentralized application (DApp) and a traditional application is that functions run on DApps are executed by an open, permissionless and decentralized network of validators. In order to provide consensus, transactions are finalized across several stages: relayed around the network, selected and executed by a miner, confirmed in a block, and accepted by other valdiators. Front-running is an attack where a malicious participant observes a transaction in an early stage of finalization and attempts to preempt it before it is finalized, typically with a competing transaction however different types of front-running attacks are possible. In this chapter, we classify front-running attacks into *insertion*, *displacement*, and *suppression* attacks, providing examples for each. While front-running is possible due to the nature of how a blockchain operates, the impact of it on a specific DApp will vary depending on the DApp's logic and any implemented mitigations. To prevent front-running, we emphasize *design practices* among the various mitigations discussed throughout this chapter. In chapter 4, we implemented a call market where timing is not a priority. Here, we provide a comprehensive evaluation of its resistance to different types of front-running attacks.

This chapter addresses the following research questions:

- **RQ5(a):** Can front-running attacks be classified according to general structures or is each attack unique in its structure? We establish a classification of attacks into three main types: displacement, insertion, and suppression. See Section 5.3.

- **RQ5(b):** Across real-world, deployed, and active DApps on Ethereum, is front-running an obscure concern that only appears in select cases, or is it prevalent across many types of DApps? We find that a wide variety of DApps are susceptible to front-running attacks, both by design and in practice. This finding is supported by [41], a paper I coauthored with Eskandari and Clark, which analyzes vulnerabilities across decentralized exchanges, crypto-collectibles, gambling services, and name services. The case studies confirm that front-running is not an isolated issue but a systemic challenge in Ethereum DApps—thereby answering **RQ5(b)**.

- **RQ5(c):** How can front-running be mitigated? We establish a classification of mitigation techniques into four main types: sequencing, confidentiality, design, and embracing it. See Section 5.4.

- **RQ5(d):** For on-chain order books and decentralized exchanges, how specifically does front-running apply and do different market mechanisms mitigate front-running more than others? We find that all market structures are susceptible to a variety of front-running attacks but the designs based call markets and batch auctions (such as Lissy) do mitigate key front-running attacks by removing time-priority from trades. See Section 5.5.

5.2 Related Work and Preliminaries

In this section, we review the foundational concepts and existing research related to front-running, both in traditional financial systems and blockchain environments, to provide a comprehensive background for understanding the subsequent analysis and proposed mitigations.

5.2.1 Traditional Front-running

Front-running is exploiting early access to non-public transaction information for personal gain. Historically, traders might overhear a broker's large order and rush to execute a trade first, profiting from temporary price changes. Alternatively, brokers might exploit their own client's orders by purchasing stock for themselves before executing the client's order. Such actions are illegal in many jurisdictions with established securities regulations.

Front-running differs from insider trading and arbitrage. It involves acting on a specific, pending transaction. Insider trading uses privileged information to predict future transactions, while arbitrage profits from reacting quickly after a trade is executed or information becomes public.

Front-running was first identified on the Chicago Board Options Exchange (CBoE)[71]. In 1977, the Securities Exchange Commission (SEC) defined it as executing options transactions based on non-public information about an imminent large transaction[2]. Initial regulations applied only to certain options markets, but were later expanded to cover all security futures [3] and further broadened in 2012 to include trading in options, derivatives, and other financial instruments [6, 5]. Front-running has also been observed in domain name trading [4, 39].

5.2.2 Front-running in Blockchain

As described in chapter 2, blockchain introduces new participants like miners who control transaction execution and order. Miners can include their own transactions without broadcasting them, committing only after starting the proof-of-work process.

Users running full nodes can see unconfirmed transactions. On Ethereum, transaction fees, known as `gas`, influence priority. Miners prioritize transactions with higher gas prices, leading to a gas auction [59]. Thus, users can front-run by submitting transactions with higher gas prices. Relaying nodes can also manipulate transaction propagation, affecting the order in which miners receive transactions [56, 70].

Research and Mitigation Strategies

Blockchain front-running relates to double-spending and rushing adversaries [66]. In double-spending, a user broadcasts a transaction to obtain an off-blockchain good or service before it is confirmed, then broadcasts a competing transaction to divert the same unspent coins back to themselves [14, 62]. Cryptographic literature models front-running by allowing a 'rushing' adversary to interact with the protocol, a concept included in simulation-based blockchain security proofs [15, 49, 66].

Aune *et al.* discuss market information leakage due to the lack of priority between transaction broadcast and miner validation, proposing a cryptographic approach similar to commit-and-reveal schemes [13]. Daian *et al.* introduce *Maximum Extractable Value (MEV)*, actions that facilitate transaction reordering or front-running for profit, and analyze its economic impact [35]. MEV initially focused on Miner Extractable Value but has since broadened to include all network participants.

5.3 A Taxonomy of Front-running Attacks

Front-running attacks can generally be categorized into a few fundamental types. By focusing on the adversary's objective rather than the specific methods employed, we identify three primary categories: *displacement*, *insertion*, and *suppression* attacks. In all scenarios, Alice is attempting to execute a function on a contract in a particular state, while Mallory aims to execute her own function on the same contract in the same state before Alice can do so. This taxonomy is a novel contribution of our research and addresses **RQ5(a)** by demonstrating that front-running attacks can be grouped into general structures rather than treated as isolated or ad hoc incidents.

Displacement Attacks. In a *displacement attack*, the adversary's goal is for Mallory's function call to be executed before Alice's, making Alice's function either ineffective or redundant. It does not matter to the adversary if Alice's function call is executed afterward or not. Typical examples include:

- Alice attempting to register a domain name, only for Mallory to register it first [61].

- Alice trying to submit a bug for a bounty, but Mallory intercepts and submits it first [23].
- Alice placing a bid in an auction, and Mallory copying and submitting the same bid first.

In these scenarios, Mallory's actions displace Alice's, preventing Alice from achieving her intended outcome.

Insertion Attacks. An *insertion attack* involves Mallory altering the state of the contract after executing her function, with the expectation that Alice's original function will run on this modified state. For instance:

- Alice places a purchase order for a blockchain asset at a price higher than the current best offer. Mallory then executes two transactions: first, she buys the asset at the best offer price, and second, she offers the asset for sale at Alice's higher purchase price. When Alice's transaction executes, Mallory profits from the price difference without holding the asset.

In this type of attack, Mallory relies on the subsequent execution of Alice's transaction to achieve her goal which is how it is distinguished from a displacement attack.

Suppression Attacks. In a *suppression attack*, Mallory's function aims to delay or prevent Alice's function from executing. Once the delay is achieved, Mallory is indifferent to whether Alice's function eventually runs. A typical examples can be:

- **Fomo3D:** In the blockchain game Fomo3D, players buy tickets to win a pot of Ether, with each ticket purchase extending the game's timer. When the timer is about to run out, Mallory deploys high gas price transactions that clog the Ethereum network, suppressing other players' attempts to buy tickets. This ensures Mallory's ticket remains the last one purchased, allowing her to win the pot.

- **Auction:** Consider an auction scheduled to end at block `#10`, where bids submitted in block `#9` are still valid. Alice plans to submit her bid in block `#9`, but Mallory preemptively floods the network with low-fee or computationally expensive transactions, delaying Alice's bid from being included in time. As a result, Alice's bid is effectively suppressed, and Mallory secures the item at a lower price or wins uncontested.

Each of these primary attacks can manifest in two variants: *asymmetric* and *bulk*. In *asymmetric attacks*, Alice and Mallory perform different operations. For instance, Alice might be attempting to cancel an offer, while Mallory tries to fulfill it first, which is known as *asymmetric displacement*. On the other hand, in *bulk attacks*, Mallory attempts to execute a large number of functions. For example, multiple users, including Alice, are trying to purchase a limited set of shares offered by a firm on a blockchain, referred to as *bulk displacement*.

5.4 Key Mitigations

In our study of front-running attacks on the blockchain, we identified several strategies to prevent, detect, or mitigate such attacks. Rather than detailing specific solutions, which may evolve over time, we outline the main principles that address these attacks. Systems may implement multiple strategies in a layered approach. This section addresses **RQ5(c)** by categorizing mitigation strategies into sequencing, confidentiality, design, and MEV-capture approaches, and assessing how each reduces front-running risk.

5.4.1 Traditional Front-running Prevention Methods

In traditional markets, front-running is often viewed as a form of insider trading and is illegal. Prevention methods include after-the-fact investigations and legal actions against offenders [44]. Measures such as dark pools [122, 29] and sealed bids [96] have been used in regulated trading systems. While these methods are not directly applicable to blockchain's decentralized and often anonymous nature, it is not inconceivable that miner extractable value (MEV) or similar behaviors could be subject to regulation. For example, miners or validators operating in jurisdictions like the USA could face compliance requirements, including restrictions on front-running. Some blockchain participants may choose to register in such jurisdictions to benefit from access to infrastructure, legal protections, or business opportunities, potentially opening the door for regulation to influence blockchain behavior.

5.4.2 Transaction Sequencing

To prevent front-running by miners, various proposals enforce rules on transaction sequencing. While FIFO sequencing presents challenges due to the distributed nature of blockchain, recent research, such as Aequitas by Kelkar*et al.*, demonstrates that achieving order-fairness in Byzantine consensus is feasible [63]. However, fair ordering has limitations, particularly in addressing suppression attacks, which it can even exacerbate. For instance, in scenarios like Fomo3D, fair ordering may simplify the execution of suppression attacks. Alternatives like pseudorandom sequencing (*e.g.,* Canonical Transaction Ordering Rule by Bitcoin Cash ABC[115, 116]) and transaction chaining offer some protection but also face practical limitations.

5.4.3 Confidentiality

Privacy-preserving techniques aim to obscure transaction details, reducing the information available to front-runners. Confidentiality can apply to various aspects of a transaction, such as the contract address, state, function calls, parameters, or sender identities. For example, in a decentralized order book, front-running can occur if adversaries observe pending buy or sell orders. By hiding key details like function calls and parameters, adversaries cannot predict trade specifics, reducing opportunities for manipulation. Additionally, obscuring the state of the order book itself (such as active orders or balances) can prevent attackers from inferring trading strategies or price-sensitive information.

Privacy-focused blockchains and tools like ZCash and AZTEC [26, 74, 78, 87, 100, 117] enhance confidentiality using cryptographic techniques, such as zero-knowledge proofs, to conceal transaction details. However, even these systems may leak residual information, such as state changes or traffic patterns, which adversaries can still exploit.

Hybrid approaches, like commit-reveal schemes, offer a promising solution for order books. During a commitment phase, users can submit encrypted orders that are revealed only after being sequenced. This ensures confidentiality while finalizing the transaction order, minimizing front-running risks. However, these approaches require careful design to prevent issues like early aborts or partial disclosures, which adversaries could exploit.

5.4.4 Embrace It

Instead of solely focusing on eliminating front-running, an alternative approach is to embrace its inevitability and desig systems to channel its economic potential constructively. By recognizing front-running as an intrinsic aspect of blockchains, we can develop mechanisms that monetize it in ways that align with the network's overall health and sustainability.

One way to do this is by leveraging Maximum Extractable Value (MEV), which refers to the maximum profit that can be extracted through transaction ordering, insertion, or censorship within a block [35]. Tools like Flashbots, a research organization dedicated to mitigating the negative externalities of MEV, provide mechanisms to democratize and optimize access to MEV opportunities.[1]

For example, Flashbots introduced MEV-Boost, an open-source software that allows Ethereum validators to outsource block building to specialized block builders.[2] Validators running MEV-Boost can maximize their rewards by accepting blocks optimized for MEV extraction. This approach aligns well with Proposer-Builder Separation (PBS), a mechanism that decentralizes block production by dividing the roles of proposing and building blocks. PBS enables specialized builders to focus on constructing high-value blocks while proposers (validators) select the most profitable block to include in the chain [21].[3]

These systems not only help mitigate the adverse effects of front-running but also turn it into a structured and economically productive activity. By embracing these tools and mechanisms, we propose a more pragmatic and incentive-aligned solution to the persistent challenges posed by front-running.

5.4.5 Design Practices

The most effective mitigation strategy is to design DApps in a way that removes any advantage gained from front-running. For example, a decentralized exchange can use a call market design instead of a time-sensitive order book [33]. In a call market, orders are executed in batches, making the arrival time of orders irrelevant and removing the financial incentive for front-running (see Chapter 4). This approach pivots the

[1]Flashbots: Bringing Transparency and Democratization to MEV, accessed Nov. 2024.
[2]MEV-Boost: An Open Source Software for Validators, accessed Nov. 2024.
[3]Ethereum Foundation, Proposer-Builder Separation (PBS), accessed Dec. 2024.

potential profits from front-running into fees collected by miners, disincentivizing the practice.

By focusing on design practices, we can create DApps that are inherently resistant to front-running. This approach not only addresses the current threats but also adapts to future developments in blockchain technology.

5.5 Evaluation of Front-running Attacks in Exchanges

As a continuation of our discussion on design practices, we now turn to the call market design (Lissy) we implemented in Chapter 4. The call market, where time is not a priority, provides an ideal framework to evaluate front-running attacks. By executing orders in batches, the call market design inherently reduces the opportunities for front-runners to gain an advantage based on transaction timing.

In the next section, we will evaluate the effectiveness of the call market design in mitigating front-running attacks (see Chapter 4 for a detailed explanation of how the call market works). We will assess its resistance to the different types of front-running attacks discussed in our taxonomy, providing a comprehensive analysis of its strengths and potential areas for improvement.

As illustrated in Table 5.1, call markets exhibit a unique resilience profile against *front-running attacks* [33, 40, 35] that differs from continuous-time markets and automated market makers. Traders are often categorized as *makers* (who add orders to a market) and *takers* (who trade against pre-existing, unexecuted orders). Continuous

			Centralized Continuous Market (Coinbase)	Partially Off-chain Continuous Market (EtherDelta)	Partially Off-chain Continuous Market w/ Roll-up (Loopring)	On-chain Continuous Market (OasisDex)	On-chain Dark Continuous Market (TEX)	On-chain Automated Market Maker (Uniswap)	On-chain Call Market w/ Price Improvement	On-chain Call Market (Lissy)	On-chain Call Market w/ Roll-up (Lissy variant)	On-chain Dark Call Market (Galal *et al.*)
Who is Mallory? Authority, Trader, Miner, Sequencer			A	A,T,M	A,T,M,S	T,M	T,M	T,M	T,M	T,M	T,M,S	T,M
Attack Example	Mallory (*maker*) squeezes in a transaction before Alice's (*taker*) order	Ins.	○	○	○	○	●	○	●	●	●	●
	Mallory (*taker*) squeezes in a transaction before Bob's (*taker 2*)	Disp.	○	○	○	○	●	○	●	●	●	●
	Mallory (*maker 1*) suppresses a better incoming order from Alice (*maker 2*) until Mallory's order is executed	Supp.	○	○	○	●	●	●	◐	◐	◐	◐
	A hybrid attack based on the above (*e.g.,* sandwich attacks, scalping)	I/S/D	○	○	○	○	●	○	○	●	●	●
	Mallory suspends the market for a period of time	Supp.	○	○	○	◐	◐	◐	◐	◐	◐	◐
	Spoofing: Mallory (*maker*) puts an order as bait, sees Alice (*taker*) tries to execute it, and cancels it first	S&D	○	○	○	○	●	○	●	●	●	●
	Cancellation Griefing: Alice (*maker*) cancels an order and Mallory (*taker*) fulfills it first	Disp.	○	○	○	○	●	○	●	●	●	●

Table 5.1: An evaluation of front-running attacks (rows) for different types of order books (columns). Front-running attacks are in three categories: Insertion, displacement, and suppression. A full dot (●) means the front-running attack is mitigated or not applicable to the order book type, a partial mitigation (◐) is awarded when the front-running attack is possible but expensive, and we give no award (○) if the attack is feasible.

markets have both types, while automated market makers have all traders as takers, with liquidity providers acting as makers. In a call market, only makers exist; trades are executed solely by submitting orders.

The front-running attacks in Table 5.1 are subcategorized, as described above, into *Insertion*, *Displacement*, and *Suppression*. This evaluation addresses **RQ5(d)** by comparing market designs and demonstrating that batch-based mechanisms like call markets offer stronger mitigation against insertion and displacement attacks than continuous markets or AMMs. In the following section, we will begin by describing each market type and its operation, followed by an in-depth analysis of each mitigation strategy.

5.5.1 Detailed Analysis of Market Types and Mitigation Strategies

Centralized Continuous Market (*e.g.,* Coinbase)

Centralized continuous markets, such as Coinbase, operate under a centralized authority that matches orders in real time. Traders submit buy and sell orders, which are executed immediately if a matching order exists. This central authority has control over the order book and transaction sequencing, which allows for efficient and fast transactions. However, this centralization introduces the risk of front-running, as the authority can potentially manipulate the order sequence.

- **Insertion Attack (Ins.): Not mitigated** ($\circ$). In a centralized exchange like Coinbase, Mallory could collude with the exchange authority to prioritize her transaction over Alice's. Additionally, if Mallory is a fast trader, she could see Alice's transaction in the mempool and add her transaction with a higher gas fee, bribing miners to execute hers first. Finally, if Mallory is the miner of the block that includes Alice's transaction, she can insert her transaction with high fidelity, making insertion attacks probabilistically successful but not guaranteed.

- **Displacement Attack (Disp.): Not mitigated** ($\circ$). The centralized control over transaction sequencing enables displacement attacks where Mallory can prioritize her transaction over Alice's. By colluding with the exchange authority or leveraging faster access to transaction data, Mallory can displace Alice's transaction, rendering it ineffective or redundant.

- **Suppression Attack (Supp.): Not mitigated** ($\circ$). The centralized authority can delay transactions, allowing suppression attacks where Mallory or the authority intentionally holds back Alice's transaction. This manipulation of transaction timing can significantly impact market dynamics.

- **Hybrid Attack (I/S/D): Not mitigated ($\circ$).** The lack of restrictions on transaction sequencing and the potential for collusion or manipulation by the centralized authority allows combination attacks involving insertion, suppression, and displacement. Mallory can leverage multiple strategies to exploit the market.

- **Market Suspension Attack (Supp.): Not mitigated ($\circ$).** The market can be suspended by the centralized authority, making it vulnerable to market suspension attacks. Mallory or the authority can halt trading activities, disrupting market operations.

- **Spoofing Attack (S&D): Not mitigated ($\circ$).** Spoofing attacks are feasible as the centralized control does not prevent placing and canceling bait orders. Mallory can manipulate market perception by placing large orders and then canceling them to influence prices.

- **Cancellation Griefing Attack (Disp.): Not mitigated** (○)**.** Cancellation griefing is possible due to the authority's control over transaction sequencing and execution. Mallory can exploit the timing of cancellation requests to fulfill orders before they are officially canceled.

Partially Off-chain Continuous Market (*e.g.,* EtherDelta)

EtherDelta represents a partially off-chain continuous market where some operations are conducted off-chain to reduce the load on the blockchain network. Orders are placed and matched off-chain, but the actual trade execution is on-chain. This hybrid approach aims to balance efficiency and decentralization. While it reduces some front-running risks, it still faces challenges due to the off-chain elements that can be manipulated.

- **Insertion Attack (Ins.): Not mitigated** (○). Since the initial handling of orders occurs off-chain, Mallory can observe pending transactions and insert her own transaction with higher gas fees to prioritize it over others when it moves on-chain. The off-chain order matching process does not prevent Mallory from front-running Alice's transaction by leveraging higher fees or faster submission.

- **Displacement Attack (Disp.): Not mitigated** (○). Displacement attacks are feasible because the off-chain order matching process does not prevent Mallory from displacing Alice's transaction, as Mallory can manipulate the timing to ensure her transaction is executed first.

- **Suppression Attack (Supp.): Not mitigated** (○). Mallory floods the network with transactions until a trader executes her order.

- **Hybrid Attack (I/S/D): Not mitigated** (○). Hybrid attacks are feasible here due to the manipulability of the off-chain order matching and on-chain execution processes. Mallory can exploit the off-chain order handling to perform complex front-running strategies.

- **Market Suspension Attack (Supp.): Not mitigated** (○). Market suspension attacks are feasible here. Mallory could disrupt the off-chain order matching process or flood the network with transactions to suspend market operations temporarily. The reliance on off-chain servers introduces vulnerabilities that can be exploited to halt the market.

- **Spoofing Attack (S&D): Not mitigated** (○). Mallory can place and cancel orders off-chain to manipulate market perception. The off-chain elements reduce the visibility of bait orders, but spoofing remains feasible as the off-chain order book can be manipulated.

- **Cancellation Griefing Attack (Disp.): Not mitigated** (○). Mallory can exploit the timing of cancellation requests and off-chain order matching to prioritize and fulfill orders before they are officially canceled. The decentralized control over transaction processing does not prevent Mallory from taking advantage of the delay between order cancellation and on-chain execution.sequencing.

Partially Off-chain Continuous Market with Roll-up (*e.g.*, Loopring)

Loopring is a partially off-chain continuous market that uses an order ring system to match multiple orders in a circular trading loop, known as an order ring. Each order ring can contain up to 16 orders, creating a loop where each order can exchange the desired tokens without needing an opposing order for its pair. This enhances scalability and reduces costs.

- **Insertion Attack (Ins.): Not mitigated** ($\circ$). Since the initial handling of orders occurs off-chain, Mallory can observe pending transactions and insert her own transaction with higher gas fees to prioritize it over others when it moves on-chain. The off-chain order ring system does not prevent Mallory from front-running Alice's transaction by leveraging higher fees or faster submission.
- **Displacement Attack (Disp.): Not mitigated** ($\circ$).The off-chain order ring system here does not prevent Mallory from displacing Alice's transaction, as Mallory can manipulate the timing to ensure her transaction is executed first.
- **Suppression Attack (Supp.): Not mitigated** ($\circ$). Mallory can flood the network with transactions until a trader executes her order.
- **Hybrid Attack (I/S/D): Not mitigated** ($\circ$). Mallory can exploit the off-chain order handling to perform complex front-running strategies.
- **Market Suspension Attack (Supp.): Not mitigated** ($\circ$). Mallory could disrupt the off-chain order ring system or flood the network with transactions

to suspend market operations temporarily.

- **Spoofing Attack (S&D): Not mitigated** (○). Mallory can place and cancel orders off-chain to manipulate market perception. The off-chain elements reduce the visibility of bait orders, but spoofing remains feasible as the off-chain order book can be manipulated.

- **Cancellation Griefing Attack (Disp.): Not mitigated** (○). Mallory can exploit the timing of cancellation requests and off-chain order ring system to prioritize and fulfill orders before they are officially canceled. The decentralized control over transaction processing does not prevent Mallory from taking advantage of the delay between order cancellation and on-chain execution.

On-chain Continuous Market (*e.g.,* OasisDEX)

OasisDEX is an example of a fully on-chain continuous market where all operations, including order placement, matching, and execution, occur on the blockchain. This full decentralization ensures transparency but makes it vulnerable to front-running attacks, as transactions are visible in the mempool before being mined, allowing attackers to manipulate the sequence.

- **Insertion Attack (Ins.): Not mitigated** (○). Since all transactions are publicly visible in the Ethereum mempool, Mallory can observe Alice's transaction and insert her own with a higher gas fee to prioritize execution.

- **Displacement Attack (Disp.): Not mitigated** (○). Mallory can displace

Alice's transaction by submitting her own transaction with a higher gas fee. The network will prioritize Mallory's transaction, allowing her to execute the trade first.

- **Suppression Attack (Supp.): Mitigated (●).** In a suppression attack, Mallory floods the network with transactions until a trader executes her order. Such selective denial of service is possible by an off-chain operator. With on-chain continuous markets like OasisDex, it is not possible to suppress Alice's transaction while also letting through a transaction from a taker—suppression applies to all Ethereum transactions or none. This decentralized nature ensures that all transactions are treated equally.

- **Hybrid Attack (I/S/D): Not mitigated (○).** The public nature of the mempool and the ability to prioritize transactions based on gas fees allow Mallory to execute complex front-running strategies.

- **Market Suspension Attack (Supp.): Partially mitigated (◐).**While it is challenging for a single actor to halt the entire market, Mallory could flood the network with transactions to temporarily disrupt trading activities. The fully on-chain nature provides some resilience but does not fully prevent temporary suspension.

- **Spoofing Attack (S&D): Not mitigated (○).** Mallory can place and cancel bait orders to manipulate market perception. The public visibility of orders makes it easy to influence the market using spoofing techniques.

- **Cancellation Griefing Attack (Disp.): Not mitigated** ($\circ$). Mallory can exploit the timing of cancellation requests to fulfill orders before they are officially canceled. The decentralized execution does not prevent Mallory from taking advantage of the delay between order cancellation and execution.

On-chain Dark Continuous Market (*e.g.,* TEX)

TEX operates as an on-chain dark continuous market, where the details of transactions are concealed until execution. This confidentiality prevents front-runners from seeing and reacting to pending transactions. By hiding transaction details and timing, FEX mitigates the risks of insertion, displacement, and suppression attacks.

- **Insertion Attack (Ins.): Mitigated** ($\bullet$). The dark pool nature conceals transaction details, preventing insertion attacks by hiding the sequence and content of transactions.

- **Displacement Attack (Disp.): Mitigated** ($\bullet$). Concealed transaction details also prevent displacement attacks, as the exact order and content are hidden.

- **Suppression Attack (Supp.): Mitigated** ($\bullet$). The dark pool prevents suppression by concealing the details and timing of transactions.

- **Hybrid Attack (I/S/D): Mitigated** ($\bullet$). The combination of concealment techniques prevents hybrid attacks by hiding the sequence and details of transactions.

- **Market Suspension Attack (Supp.): Partially mitigated (◐).** While the dark pool conceals transaction details, making selective suspension more difficult, a determined attacker could still flood the network to attempt a market-wide suspension.

- **Spoofing Attack (S&D): Mitigated (●).** Spoofing is not feasible due to the concealment of transaction details and timing.

- **Cancellation Griefing Attack (Disp.): Mitigated (●).** Concealment of transaction details prevents cancellation griefing by hiding the sequence and content.

On-chain Automated Market Maker (*e.g.,* Uniswap)

Uniswap, an automated market maker (AMM), uses a mathematical formula to determine prices and match trades. Liquidity providers supply tokens to the AMM, and traders execute trades against this liquidity. This model processes transactions continuously and automatically, preventing front-running to some extent. However, it is still vulnerable to specific attacks like sandwich attacks, where attackers manipulate the sequence of transactions.

- **Insertion Attack (Ins.): Not mitigated (○).** Insertion attacks are feasible in Uniswap. Although the AMM model processes transactions based on a deterministic formula, front-runners can still observe pending transactions in the mempool and attempt to insert their own transactions with higher gas fees to be processed first.

- **Displacement Attack (Disp.): Not mitigated (○).** Displacement attacks are possible as attackers can prioritize their transactions over others by paying higher gas fees. This allows front-runners to execute trades before others, taking advantage of favourable prices.

- **Suppression Attack (Supp.): Mitigated (●).** Suppression is not feasible in the Uniswap AMM model because all transactions are processed continuously and automatically. There is no mechanism for selectively delaying or suppressing specific transactions.

- **Hybrid Attack (I/S/D): Not mitigated (○).** Hybrid attacks, which involve a mix of insertion, displacement, and suppression, are feasible due to the possibility of insertion and displacement attacks in Uniswap's AMM model.

- **Market Suspension Attack (Supp.): Partially mitigated (◐).** Market suspension is partially mitigated in Uniswap. While the continuous and automatic processing of transactions makes it difficult to suspend the market entirely, a determined attacker could potentially flood the network with transactions to temporarily disrupt the market.

- **Spoofing Attack (S&D): Not mitigated (○).** Spoofing is possible in Uniswap. Attackers can place and cancel bait orders to manipulate the perception of the market, even though transactions are processed based on a formula.

- **Cancellation Griefing Attack (Disp.): Not mitigated** (○). Cancellation griefing attacks are possible because attackers can exploit the ability to prioritize and cancel transactions by adjusting gas fees.

On-chain Call Market with Uniform Price Improvement

In a traditional call market, a market clearing price is chosen and all trades are executed at this price. All bids made at a higher price will receive the assets for the lower clearing price (and conversely for lower ask prices): this is called a *price improvement* and it allows traders to submit at their best price.

- **Insertion Attack (Ins.): Mitigated (●).** The call market batches orders and executes them at a uniform price, removing the advantage of timing-based attacks and making insertion attacks infeasible.

- **Displacement Attack (Disp.): Mitigated (●).** Uniform price execution prevents displacement attacks by ensuring all orders are executed at the same price, regardless of the submission time.

- **Suppression Attack (Supp.): Partially mitigated (◐).** While the call market processes all orders in batches at a predetermined time, a determined attacker could attempt to flood the network with transactions to delay the inclusion of specific orders. However, because all orders are executed simultaneously, the effectiveness of such suppression is limited. We still award a call market partial mitigation since suppression attacks are expensive (as evidenced by theFomo3D attack in [40]). If the aim of suppression is a temporary denial of service (captured by attack 5 in the table), then all on-chain markets are vulnerable to this expensive attack.

- **Hybrid Attack (I/S/D): Not mitigated (○)**. Some attacks combine more than one insertion, displacement, and/or suppression attack. A hybrid front-running attack allows Mallory to extract any price improvements. Consider the case where Alice's ask crosses Bob's bid with a material price improvement. Mallory inserts a bid at Alice's price, suppresses Bob's bid until the next call, and places an ask at Bob's price. She buys and then immediately sells the asset and nets the price improvement as arbitrage.

- **Market Suspension Attack (Supp.): Partially mitigated (◐)**. While the call market's batched processing reduces the risk of market suspension by making it harder for a single actor to halt the market, a determined attacker could still flood the network with a high volume of transactions to disrupt the market temporarily. This makes complete suspension difficult but not impossible.

- **Spoofing Attack (S&D): Mitigated (●)**. Spoofing is prevented by the call market's structure, which batches and processes orders uniformly, making bait orders ineffective.

- **Cancellation Griefing Attack (Disp.): Mitigated (●)**. In a call market with uniform price execution, all trades are executed at a single clearing price at the end of the batch period. This makes it harder for a front-runner to prioritize their transaction after a cancellation request.

On-chain Call Market (*e.g.,* Lissy)

For a detailed explanation of how Lissy works, see Chapter 4.

- **Insertion Attack (Ins.): Mitigated (●).** The call market batches orders and executes them at a uniform price, removing the advantage of timing-based attacks and making insertion attacks infeasible.

- **Displacement Attack (Disp.): Mitigated (●).** Uniform price execution prevents displacement attacks by ensuring all orders are executed at the same price, regardless of the submission time.

- **Suppression Attack (Supp.): Partially mitigated (◐).** While the call market processes all orders in batches at a predetermined time, a determined attacker could attempt to flood the network with transactions to delay the inclusion of specific orders. However, because all orders are executed simultaneously, the effectiveness of such suppression is limited.

- **Hybrid Attack (I/S/D): Mitigated (●).** To mitigate this in Lissy, all price improvements are given to the miner (using `block.coinbase.transfer()`). This does not actively hurt traders—they always receive the same price that they quote in their orders—and it removes any incentive for miners to front-run these profits.

- **Market Suspension Attack (Supp.): Partially mitigated (◐).** While the call market's batched processing reduces the risk of market suspension by making it harder for a single actor to halt the market, a determined attacker could still flood the network with a high volume of transactions to disrupt the market temporarily. This makes complete suspension difficult but not impossible.

- **Spoofing Attack (S&D): Mitigated (●).** Spoofing is prevented by the call market's structure, which batches and processes orders uniformly, making bait orders ineffective.

- **Cancellation Griefing Attack (Disp.): Mitigated (●).** Lissy mitigates these attacks by running short-lived markets with no cancellations.

On-chain Call Market with Roll-up (*e.g.,* Lissy variant on *Arbitrum*)

In our Lissy variant on *Arbitrum*, traders can submit transactions to the Layer 1 Inbox contract instead of directly to the Lissy DApp. This has the same front-running profile as Lissy itself; only the Layer 1 destination address is different. If a sequencer is mandatory, it acts with the same privilege as a Layer 1 Ethereum miner in ordering the transactions it receives. Technically, sequencers are not limited to roll-ups and could be used in the context of normal Layer 1 DApps, but they are more apparent in the context of roll-ups. A sequencer could be trusted to execute transactions in the order it receives them, outsource to a fair ordering service, or (in a tacit acknowledge of the difficulties of preventing front-running) auction off permission to order transactions to the highest bidder (called a *MEV auction*). As shown in Table 5.1, a sequencer is an additional front-running actor but does not otherwise change the kinds of attacks that are possible.

On-chain Dark Call Market(*e.g.,* Galal *et al.*)

On-chain dark call markets aim to enhance transaction confidentiality by hiding the details of orders until execution. This is designed to reduce the risk of front-running by preventing attackers from gaining insight into pending transactions. While this approach increases privacy, it does not make dark call markets inherently more resilient to front-running compared to lit (transparent) markets. However, the added confidentiality can provide some protection against predatory trading algorithms that react quickly to public trades without directly front-running.

- **Insertion Attack (Ins.): Mitigated (●)**. The confidentiality provided by dark call markets hides the details of pending transactions, making it difficult for attackers to insert their own transactions before others. Without visibility into the order book, attackers cannot easily prioritize their transactions to front-run others.

- **Displacement Attack (Disp.): Mitigated (●)**. Similar to insertion attacks, the concealed nature of orders in dark call markets prevents attackers from displacing other transactions. Since the attackers do not know the content and sequence of orders, they cannot manipulate the order execution in their favor.

- **Suppression Attack (Supp.): Partially mitigated (◐)**. A determined attacker could still flood the network with transactions to attempt to delay the processing of certain orders, making complete prevention difficult.

- **Hybrid Attack (I/S/D): Mitigated (●).** These attacks are fully mitigated by the confidentiality of dark call markets. Since the order details are hidden until execution, attackers cannot combine these attack vectors effectively to exploit the market.

- **Market Suspension Attack (Supp.): Partially mitigated (◐).** While dark call markets hide transaction details, making selective suspension more difficult, a determined attacker could still flood the network with transactions to attempt a market-wide suspension. The confidentiality layer complicates such attacks but does not fully prevent them.

- **Spoofing Attack (S&D): Mitigated (●).** Spoofing is less effective in dark call markets because the details of orders are concealed. Attackers cannot place and cancel bait orders to manipulate the market perception as easily since other traders do not see these orders.

- **Cancellation Griefing Attack (Disp.): Mitigated (●).** The confidentiality of orders in dark call markets prevents front-runners from exploiting cancellations. Since the cancellation and the new order details are hidden, attackers cannot easily prioritize their transactions to benefit from the canceled orders.

5.6 Discussion

There are two main takeaways from Table 5.1. Call markets provide strong resilience to front-running only bested slightly by dark markets like TEX [64], which leverage advanced cryptographic techniques to enhance transaction confidentiality. However, call markets achieve their resilience through design, without relying on cryptography or two-round protocols. Additionally, dark call markets, like those proposed by Galal *et al.* [48], offer no more resilience to front-running than lit markets, although their confidentiality features can protect against predatory trading algorithms that react quickly to public trades without actually front-running.

5.7 Concluding Remarks

In this Chapter, we explored the multifaceted challenge of front-running attacks in blockchain systems, categorizing them into three main types: displacement, insertion, and suppression, thereby offering a structured framework to better understand and address these vulnerabilities.

We addressed key questions surrounding front-running: the classification of attack structures, confirming that displacement, insertion, and suppression attacks comprehensively capture the fundamental patterns of front-running (**RQ5(a)**); assessing its prevalence across Ethereum DApps and showing its systemic nature across diverse categories, such as decentralized exchanges, crypto-collectibles, and gambling services (**RQ5(b)**); evaluating mitigation strategies through transaction sequencing, confidentiality techniques, and DApp design (**RQ5(c)**); and analyzing the effectiveness of market mechanisms like call markets and batch auctions in mitigating front-running on on-chain order books (**RQ5(d)**).

A central focus was on the call market mechanism, discussed in detail in Chapter 4. Call markets effectively remove time-priority from trades, reducing susceptibility to key front-running attacks such as insertion and displacement, though they remain partially vulnerable to suppression due to their operational complexity. This evaluation, summarized in Table 5.1, highlights the potential of innovative market designs in creating more robust decentralized systems.

As blockchain technology continues to evolve, thoughtful DApp design and innovative market mechanisms offer a promising path to mitigate front-running while preserving the principles of decentralization and transparency. This chapter emphasizes the need for continued research into both technical and economic strategies to address these persistent challenges in blockchain ecosystems.

Chapter 6

Demystifying Stablecoins

This chapter is based on the paper "SoK: Demystifying Stablecoins" published in Communications of the ACM, 2020. This paper was co-authored by Jeremy Clark and Didem Demirag.

6.1 Introduction

While the previous chapters focused on transactional efficiency, market design, and adversarial risks in DeFi, this chapter turns to the foundational issue of price stability, which underpins the usability and accessibility of DeFi systems.

The preceding chapters of this dissertation examined fast exits from rollups (Chapter 3), decentralized order books (Chapter 4), and front-running vulnerabilities (Chapter 5) – all of which highlight key infrastructure and market-layer concerns in decentralized finance. However, for any financial system to function effectively, especially from a user perspective, stable value representation is essential. This chapter addresses this critical dimension by exploring stablecoins, which serve as the backbone of pricing, settlement, and risk mitigation in DeFi. By categorizing and evaluating the core mechanisms behind stablecoins, this chapter complements the broader theme of user-centric DeFi design by considering not just how users interact with DeFi systems, but also the reliability of the value they transact in.

The first wave of cryptocurrencies, starting in the 1980s, attempted to provide a digitization of government-issued currency (or 'fiat currency' as cryptocurrency enthusiasts say) [86]. The second wave, represented prominently by Bitcoin [83], provide their own separate currency — issued and operated independently of any existing currencies, governments, or financial institutions. Bitcoin's currency (BTC) is issued in fixed quantities according to a hardcoded schedule in the protocol.

In the words of Bitcoin's pseudonymous inventor, "*there is nobody to act as a central bank... to adjust the money supply... that would have required a trusted party to determine the value because I don't know a way for software to know the real world value of things. If there was some clever way, or if we wanted to trust someone to actively manage the money supply to peg it to something, the rules could have been programmed for that. In this sense, it's more typical of a precious metal. Instead of the supply changing to keep the value the same, the supply is predetermined and the value changes*" [84].

This chapter addresses the following research questions:

- **RQ6(a):** Can stability mechanisms for stablecoins be systematically classified based on core structural elements? Yes! Stablecoins can be classified into two main categories: *backed* and *intervention-based*. *Backed* stablecoins are those whose value is supported by underlying assets or collateral. These assets may include fiat currencies, cryptocurrencies, or other reserves, and their mechanisms can involve direct redemption guarantees or indirect collateralization strategies. *Intervention-based* stablecoins, on the other hand, rely on algorithmic or active adjustments, such as modifying the token supply or transferring assets, to stabilize their value without requiring traditional collateral. These interventions are designed to respond dynamically to market conditions. See Table 6.1.

- **RQ6(b):** How do the risk and security profiles of the different stablecoin mechanisms compare to each other and to traditional forms of digital cash? There are significant differences in price stability, trust assumptions, centralization risks, potential reserve mismanagement, over-collateralization requirements, reliance on oracles, use of heuristics, and privacy. See Table 6.3 for a complete comparison between these mechanisms.

- **RQ6(c)**: Do intervention-based stablecoins effectively achieve stability, or do their inherent risks and limitations undermine their viability? Intervention-based stablecoins strive for stability but face risks like feedback loops and reliance on heuristic models, which can fail under stress. Examples, as discussed in Section 6.6.1, highlight how these flaws undermine their viability.

- **RQ6(d)**: Can gas-backed stablecoins feasibly provide stability and serve as a

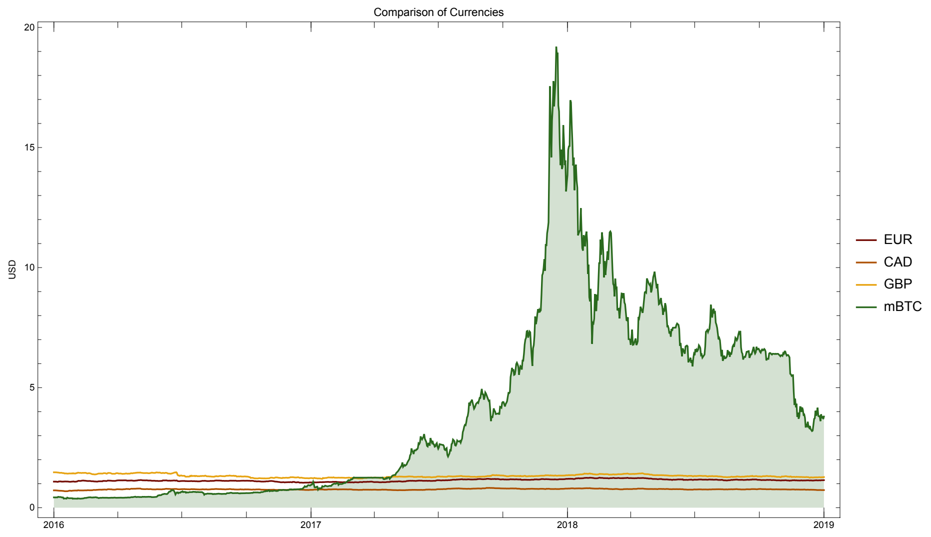


Figure 6.1: Comparison among fiat currencies and Bitcoin: The values are retrieved daily between 01 Jan 2016 and 01 Jan 2019. Note that 1000 mBTC = 1 BTC.

practical unit of account within blockchain ecosystems? Gas-backed stablecoins show potential for stability by anchoring value to network usage, but their feasibility depends on user adoption, predictable gas pricing, and robust tokenization mechanisms. Based on our measurements, gas is not a suitable stablecoin. See Section 6.6.4 for a thorough discussion on this topic.

Without active management, the exchange rate of BTC with governmental currencies has been marked by extreme volatility (see Figure 6.1). Squint at the chart to notice how the GBP drops around June 2016: this mild-looking pinch is actually the 'sharp decline' and 'severe swing' that followed the Brexit referendum in the UK. However, it is completely overshadowed when placed beside BTC's large fluctuations.

A third wave? Extreme volatility is not specific to Bitcoin (BTC), and can also be seen in its contemporaries Ethereum (ETH) and Ripple (XRP). This instability is an issue of practical importance: volatility encourages users to hoard (if it is going up) or avoid (if it is going down) the currency rather than use it. It makes lending risky, as currency movements can exceed interest payments. A lack of lending and credit inhibits the formation of mature financial markets. In response, a flood of proposals have been made for new cryptocurrency designs that purport to provide a stable exchange rate similar to (or exactly mirroring) a government-issued currency like the USD. These designs are called stablecoins.

Stablecoins have garnered a lot of attention recently, both positive and negative. According to *CoinMarketCap*, more value in Tether changes hands across a given day than Bitcoin. This despite questions about Tether's reserves and regulatory investigations into its affiliates. The announcement of Facebook's Libra made international headlines and has been remarked on by the Fed, US legislators, and the even the sitting President. Another project, Basis (*née* Basecoin) raised $133M in Venture Capital but folded up when it could not find a tenable path through US financial regulations. Central banks, including those of Sweden and Denmark, have explored the idea of government-issued stable cryptocurrencies.

In this Chapter, we aim to clarify the mechanics of stablecoins. While most projects provide white papers detailing their designs and numerous online articles survey various approaches, organizing this information systematically is challenging. Many white papers are laden with jargon—terms are often undefined or used inconsistently with other projects and financial literature. Furthermore, system components are frequently mislabeled. For example, elements that meet the definitions of securities or derivatives might be incorrectly labeled as bonds or loans. This could be due to a lack of precision, attempts to make unconventional protocols seem more traditional, or efforts to avoid regulatory scrutiny.

To address these issues, we have made a concentrated effort to offer clear and direct explanations (see Table 6.2). Additionally, we acknowledge the work of other academics who have developed their own taxonomies [93, 79]. Our goal is to provide a comprehensive understanding of stablecoin mechanics that overcomes the common pitfalls found in existing literature.

6.2 Related Work and Preliminaries

An early idea for a stable cryptocurrency was introduced by Ametrano in 2016 [10] which tweaked supply (see Section 6.6.3) In 2017, Robert Sams introduced a new dual token system that uses two coins to achieve price stability: (i) a stablecoin (*e.g.,* fiat currency) and (ii) a volatile coin (*e.g.,* equity shares) [99] (see Section 6.6.3). These informed the designs of stablecoins we discuss in the paper.

There are many blog posts and professional reports providing an overview of stablecoins with essentially the same goal as this paper. Most have the same general categorization: (i) fiat-collateralized, (ii) crypto-collateralized and (iii) non-collateralized, also known as algorithmic,*e.g.,* [8], [111], [9]. Buterin, in one of the earliest blog posts on stable cryptocurrencies, discussed different techniques to measure cryptocurrencies' price and how to make adjustments in the supply to achieve a fixed price accordingly [42]. Bitmex looked at the mechanics of the distributed stablecoins while focusing on two case studies (*i.e.,* BitShares (BitUSD) and MakerDAO (Dai)) [7]. Another report by crypto company Blockchain provides an extensive classification of 57 stablecoins together with discussions on issues related to governance (*e.g.,* legal structure, investors, partners *etc.*) [18]. They extend this research in a newly published report that reflects a significant update on their previous work [110]. In one of its blog posts, Consensys [1] describes stablecoins as “crypto-assets that maintain a stable value against a target price (*e.g.,* USD)” and classify them according to three main categories [103]. In [89], the authors use a slightly different categorization to group 13 stablecoin projects into two broad categories: (i) centralized– which itself contains subcategories based on the type of the asset the coins are backed by (*e.g.,* fiat and gold), (ii) decentralized. Our survey can be differentiated from the related work in its focus: we strive for clear descriptions of the mechanics of stability within a coin, without glossing over details or substituting financial jargon for explanation. We focus less on any infrastructure around a stablecoin deployment (*e.g.,* its code, interactions between contracts, governance, *etc.*).

6.2.1 Prices

The price of a cryptocurrency, like any asset, is commonly quoted as the most recent exchange rate between it and another asset, such as USD. For example, if 1 BTC is worth \$3598.76 USD, this indicates that two participants recently exchanged BTC

[1]https://consensys.net

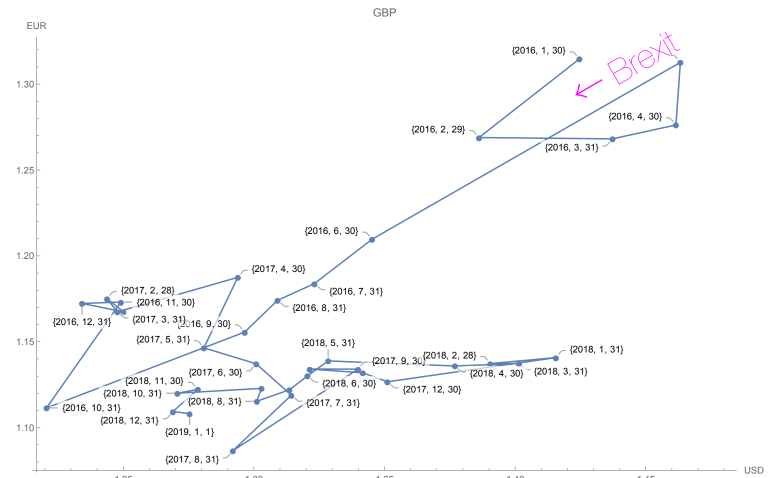
and USD at this valuation. However, this price is not fixed—it may not represent the value of the next trade and becomes less reliable as time passes between trades.

Instead of relying solely on the last sale price, an asset's value is better understood through its **bid price** (the highest price someone is willing to pay) and **ask price** (the lowest price someone is willing to sell for). The difference between these is called the **bid-ask spread**, and the actual trade price will fall between these values. For stablecoins designed to maintain a $1 USD value, stability requires ensuring that (1) the bid price does not exceed $1 USD and (2) the ask price does not fall below $1 USD. Deviations, however, are natural due to transaction friction and market dynamics.

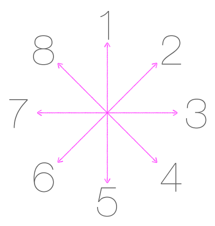
6.2.2 Exchange Rates

Consider that several hours after writing the previous section, 1 BTC is now priced at $3566.56 USD. In one sense, the price of BTC decreased by $32.20. However it is exactly equivalent to say the price of $1 USD increased by 0.002 mBTC. This raises a natural question: did BTC decrease in price or did USD increase in price? With an exchange rate, it is impossible to tell. We only know that the price of BTC and USD became closer in price over this short period of time.

To determine which currency is moving, one might consider a third or forth currencies (*cf.* US Dollar Index) to try and triangulate if BTC is moving in price, or USD is moving in price, or both. For example, in Figure 6.1 it certainly appears that BTC is the currency that is moving because the rest of the currencies are stable relative to each other. The only alternative is that USD, EUR, GBP, and CAD are



(a) GBP with respect to EUR and USD.



(b) Direction labels.

Direction	**Interpretation**
1/5	Y is losing (1) / gaining (5) value
2/6	Plotted asset is gaining (2) / losing (6) value
3/7	X is losing (3) / gaining (7) value
4/8	Plotted asset is gaining (4) / losing (8) value against X, while losing (4) / gaining (8) value against Y

(c) The simpliest interpretation of the plots where X refers to the currency on the x-axis (likewise Y).

Figure 6.2: A connected scatter plot of GBP's exchange rate with EUR and USD demonstrating the effect of Brexit on GBP. Supporting documentation helps interpret the line movements in the plot.

volatile currencies that move together as a cluster relative to the stability of BTC. But it is much simpler to conclude that BTC is moving.

In order to apply this same logic in a visual way, we have created a number of charts like the one provided in Figure 6.2a. Unlike most exchange rate graphs, these do not use a time axis. Instead each axis is a reference currency. In this case, the price of GBP (plotted value) in USD (x-axis) and EUR (y-axis) form a coordinate. For the last day of each month, a new coordinate is added and joined with a line from the previous value. This is inspired by similar charts on the website FiveThiryEight for things like kicking distance in football[2] and they have been called connected scatter plots.

Lines in a connected scatter plot can move in any direction. Figure 6.2b shows how we number the directions from 1 (upward or due north) clockwise to 8 (northwest). For each direction, we describe the simplest interpretation of what that price direction means. By simplest, we mean specifically that we keep an explanation that involves a single currency moving rather than an explanation that involves a pair of currencies moving in tandem. For example, in Figure 6.2a, GBP shows a drastic movement along direction 6 starting at the time period marked Brexit. This means that GBP is losing value against both EUR and USD. The simplest explanation is that the movements are originating from GBP which is consistent with it losing value after Brexit. Later, GBP shows a lot of horizontal movements along the 7/3 line. The simplest explanation for this segment is volatility in USD rather than GBP. A copy of the datasets and codes of all the charts can be found on our GitHub repository.[3]

We will return to these charts later in Section 6.6.3 where we will use a government currency as one reference (USD on the x-axis) and a cryptocurrency as the other reference (BTC on the y-axis). A stablecoin should exhibit mostly vertical movements along the 1/5 direction.

[2] "The 52 Best—And Weirdest—Charts We Made In 2016," *FiveThirtyEight*, 30 Dec 2016.
[3] https://github.com/mahsamoosavi/Stablecoins.

6.2.3 Valuation

The price of an asset reflects recent trading activity but may not represent its fundamental value. For example, BTC's price might be linked to speculative factors or operational costs, like electricity consumption for mining. Unlike stocks, which have a book value based on a firm's financials, cryptocurrencies lack a definitive fundamental valuation framework, leaving their prices more susceptible to market sentiment and volatility.

6.2.4 Stability and Volatility

Price fluctuations in a currency can stem from fundamental changes, such as new information about its value, or from transient market volatility caused by uninformed trading. Stablecoins aim to mitigate volatility by introducing mechanisms to maintain a fixed exchange rate, often defined as pegging to a specific fiat currency like USD. This reduces the speculative nature of the currency and facilitates its use as a reliable medium of exchange, unit of account, and store of value.

6.2.5 Functions of Money and Lending

For a currency to be effective, it must function as a **medium of exchange**, **unit of account**, and **store of value**. Cryptocurrencies like Bitcoin face challenges in these areas due to their volatility. Stablecoins, by design, aim to resolve these challenges by providing a stable store of value. This stability fosters the development of low-risk lending markets, which are foundational to modern economies. Stablecoins also enable lending by mitigating risks associated with currency depreciation or appreciation, thus unlocking the potential for more mature financial systems.

6.3 How Do Stablecoins Work?

We started by finding stablecoin projects on *CoinDesk*, an online news source for cryptocurrencies, using search queries like stablecoins," stability," and "price-stable." We read 185 articles up to January 11, 2019.[4] For the 25 projects for which we could find sufficient documentation, we classified them in Table 6.1. This classification is done according to what the projects assert they do—we provide no warranty of what the projects do in reality. We sort projects according to their rank on *CoinMarket-Cap*, which ranks cryptocurrencies that are actively traded on an exchange service. Unlisted projects are ranked $\perp$. Since 2019, many new stablecoins have been introduced, but they continue to follow the same core design patterns and fit within the taxonomy presented here.

Next, we distilled each proposal into a core stability mechanism. Instead of enumerating the intricate details of how each 'brand' of stablecoin works—details that could change tomorrow—we concentrate on communicating the fundamentals. Broadly, the proposals can be split into two categories: (1) ones that try to directly match the stability of a second asset, such as the USD, and could not exist without this underlying asset, and (2) ones that propose independent currencies with algorithmic and/or human intervention mechanisms for providing stability. This directly

[4]Given its high profile, we also include Facebook's Libra Coin which was released after this date.

Class	**Mechanism**	**Resembles**	Rank
Backed	Directly-Backed & Redeemable†	USDC	20
		TrueUSD	26
		Paxos	38
		Gemini Dollar	52
		StableUSD (USDS)	685
		Stronghold USD	891
		Petro	1210
		Libra Coin, Ekon, WBTC, emparta	$\perp$
	Directly-Backed	Tether	6
		EURSToken	95
		BitCNY	304
		Terracoin	1280
		Saga	1495
		GJY, Novatti AUD, UP-USD	$\perp$
	Indirectly-Backed	Dai	57
		BitUSD	398
		Nomin	$\perp$
Intervention	Money Supply Adjustments	Ampleforth	$\perp$
		RSCoin	$\perp$
	Asset Transfer	NuBits	892
		CarbonUSD	1262
		Basecoin	$\perp$

Table 6.1: Stablecoin proposals as of January 11, 2019. † *Disclaimer:* Projects are classified according to what they assert; *e.g.,* we provide no warranty that projects classified as 'redeemable' provide actual redemption of the assets that back their coins. Rank corresponds to *CoinMarketCap*.

addresses **RQ6(a)**, which asks whether stability mechanisms can be systematically categorized based on core structural elements. More detail is provided in Table 6.2.

Stability Mechanisms
Directly Backed and Redeemable. Alice is a trusted third party and uses Ethereum to instantiate a decentralized application (DApp) which issues 1000 AliceCoins as standard tokens (*e.g.,* ERC20). She asks \$1 USD for 1 AliceCoin and promises to redeem any AliceCoin for \$1 USD. If Bob buys 10 AliceCoins for \$10 USD, Alice deposits the \$10 USD in a bank account. Any time Alice receives a buy order for AliceCoins and does not have any left to sell, she creates new ones to sell. If Carol wants to redeem 5 AliceCoins, Alice withdraws \$5 USD and exchanges it with Carol, taking those AliceCoins out of circulation. Alice frequently publishes bank statements showing that her account holds enough USD to redeem all coins in circulation (the number of AliceCoins can be checked anytime on Ethereum).
Directly Backed. Again, Alice is a trusted third party that issues 1000 AliceCoins as ERC20 tokens. She asks \$1 USD for 1 AliceCoin and promises to deposit and hold the payment in a bank account. As before, Alice creates new AliceCoins when she runs out and publishes frequent bank statements. Unlike above, she offers no direct redemption of AliceCoins for USD.
Indirectly Backed. Alice is no longer assumed to be trustworthy. She sets up a DApp that can hold ETH and issue tokens. The DApp determines how much ETH is equivalent to \$1.50 USD using the current exchange rate, provided to the DApp by a trusted third party oracle, and Alice deposits this amount of ETH into the DApp. The DApp issues to Alice two places in a line — each place is a transferrable token. At some future time, the holder of the first place in line can redeem up to \$1.00 USD worth of the deposited ETH at the going exchange rate, and the holder of the second place in line gets any remaining ETH. Alice will transfer the first place in line (as a stable coin called AliceCoin) to Bob for \$1.00 USD, and will hold or sell the second place in line. When Bob redeems the AliceCoin, it will be worth \$1 USD in ETH when the entire deposit of ETH is worth more than \$1 USD. If the exchange rate drops enough, the entire deposit will be worth less than \$1 USD — Bob will get all of the deposit and the holder of the second place in line will get nothing.
Money Supply Adjustments. Alice forks Bitcoin to create a new altcoin called AliceCoin. She tweaks the schedule for releasing new AliceCoins (called the coinbase amount in Bitcoin) according to the rules outlined below. She sets up a trusted oracle for the latest exchange rate of AliceCoins to USD. AliceCoin is programmed to apply an intervention when the price of an AliceCoin exceeds \$1.02 USD or dips below \$0.98 USD. If the price exceeds \$1.02 USD, the miner is allowed to increase the coinbase amount (the amount is determined by some mathematical relationship with how much the price exceeds \$1.02 USD). If the price dips under \$0.98 USD, the miner must decrease the coinbase amount based on the same relationship. The correctness of the claimed coinbase is verified by other miners in deciding to accept or reject a mined block, as per all other checked conditions in Bitcoin.
Asset Transfer. Alice instantiates a DApp with an ERC20 token called AliceCoin. The DApp is programmed to apply an intervention when the price of an AliceCoin exceeds \$1.02 USD or dips below \$0.98 USD according to a trusted oracle. If the price exceeds \$1.02 USD, the DApp creates new a set of AliceCoins (as above, according to some mathematical relationship) and transfers them to users waiting in line for them. How do users wait in line? When the price dips under \$0.98 USD, the DApp creates new positions at the end of the line and auctions them off to the highest bidder. The payment for a place in line is made in AliceCoins from the bidder to the DApp and the DApp destroys the payment. The place in line is a transferrable token. If the line is empty, AliceCoins are distributed according to a fallback policy (see main text).

Table 6.2: Major types of stability mechanisms for stablecoins

Mechanism	Price		Trust				Target		
	Corrects undervaluation	Corrects overvaluation	Decentralizes issuance	Decentralizes redemption	Decentralizes transfer	No trusted oracle	Any asset	Any price level	Any event
Traditional Digital Cash	●	●				●	●		
Traditional Cryptocurrency			●	×	●	●			
Directly Backed and Redeemable	●	●			●	●	●		
Directly Backed		●			●	●	●		
Indirectly Backed	◐	●	●	●	●		●	●	
Money Supply Adjustments	?	◐	●	×	●	◐	●	●	●
Asset Transfer	?	◐	●	×	●	◐	●	●	●

Table 6.3: Comparative evaluation of mechanisms to design stablecoins: ● indicates the properties (columns) are fulfilled by the corresponding mechanism (rows) within reason, ◐ means the property is fulfilled but the fulfillment is bounded, ? indicates a heuristic has been proposed for stability and the conditions under which it will work are not well-established enough to evaluate, and × indicates the property is not applicable. Price, trust, and target are explained in Section **??**.

6.4 Type 1: Backed Stablecoins

6.4.1 Directly-Backed and Redeemable

For stablecoins in this category, the firm operating the currency will obtain a reserve of some valuable asset—it might be USD or another sovereign currency, gold or another commodity, or a basket of multiple assets. It will then issue digital tokens that represent a unit of the underlying asset (to illustrate, assume a token is redeemable for 1 USD) which can be exchanged online.

This idea predates Bitcoin: Liberty Reserve provided a similar digital currency, with some caveats about its redeemability (not to mention its legality). However Liberty Reserve, e-gold, and similar pre-blockchain services would maintain transaction details and account balances on a private server. Blockchain enables decentralized trust for the transactions, while the coin creation and redemption processes rely on a trustworthy firm. In short, this type of stablecoin is more centralized than Bitcoin but less than Liberty Reserve. For analysis, we need a finer grained approach to trust assumptions which Table 6.3 tries to capture. Also consider that while decreasing centralization can be good for trust and transparency, additional measures are needed to ensure it is not harmful for privacy.

Recall the mechanism for issuing AliceCoins in Table 6.2. If buyers are willing to pay more than \$1 USD for 1 AliceCoin, new coins can be generated for \$1 USD and sold to these buyers for a profit, ensuring bids return to \$1 USD (it corrects overvaluation). If sellers are willing to take less than \$1 USD for 1 AliceCoin, these coins can be bought and redeemed for a profit, ensuring offers return to \$1 USD (it corrects undervaluation). In reality, transactions are not free, efficient, or entirely frictionless and some price deviation is expected. If redemption is ever in doubt, then the price can fall freely from \$1 USD (although this will not necessary happen, see next section). The trustworthiness of the operating firm and the custodian of the reserves is essential, and financial audits are an important step to establishing confidence (although many pitfalls exist when auditing blockchain-based assets [95]).

6.4.2 Directly-Backed

What if a stablecoin operated exactly as in the previous section but did not offer a redemption process for the coin's underlying assets? If we could not find a clear assertion of redemption, we listed the project under this category in Table 6.1.

Recall the mechanism in Table 6.2. Bids will not exceed \$1 for the same reason as the previous section. However there is no longer a way to profit if offers vary between \$0 USD to \$1 USD (*i.e.,* the mechanism does not prevent undervaluation). Generally coins in this category are in fact 'redeemable' by one user: the firm operating the coin. It could purchase undervalued coins to release \$1 USD from its reserves. For this reason, stablecoins in this category are scrutinized (to the extent made possible by the firm) to ensure reserves are intact. If every AliceCoin was not backed by \$1 USD, Alice could overissue AliceCoins to enrich herself.

The largest coin is this category is Tether. Tether claims to be redeemable, but the redemption process is reported by users to have a lot of friction, it is accused of issuing coins to manipulate markets [51], and it has not always maintained full reserves of USD to allow all Tether to be redeemed (for these reasons, we categorize it here). To many, it is a mystery why Tether remains highly liquid with daily trading volumes exceeding all other cryptocurrencies in value (according to *CoinMarketCap* at the time of writing). One explanation is that it is too useful to fail.

A key use-case, illustrated by Tether and the affiliated exchange Bitfinex, is as a temporary store of value for traders and speculators. A trader that wants to divest their BTC for USD has three options. She can (1) hold the USD in her exchange account, which can be used only on the same exchange and requires the exchange to be a trustworthy custodian. She can (2) withdraw the USD from the exchange but this requires identity verification (in most jurisdictions), a bank that will accept proceeds of cryptocurrency trading, and a substantial time delay. A balanced alternative is to (3) exchange BTC into a stablecoin which can be withdrawn from the exchange (*i.e.,* moved from the exchange to Alice's private key) with little friction, delay, or regulatory oversight. The withdrawn stablecoin can be moved onto a different exchange, transferred to other users, or used for direct purchases without involving the original exchange. In short, it offers more flexibility than leaving USD in an exchange account and less friction than withdrawing USD.

6.4.3 Indirectly-Backed

Both of the previous mechanisms placed heavy trust assumptions on the firm operating the currency. Could a currency be managed autonomously by a DApp? The key idea of this mechanism is to offer a redeemable token that can be converted into $1 USD worth of ETH at the going USD/ETH exchange rate. Therefore the amount of ETH received ETH will grow or shrink depending on the exchange rate. Because a blockchain has no inherent knowledge of exchange rates, this mechanism still requires one trustworthy entity called an oracle to write the exchange rate into the blockchain (or consensus can be taken across a set of oracles).

Recall the mechanism in Table 6.2. Bids for an AliceCoin in excess of $1 USD will be fulfilled as long as there are individuals like Alice willing to lock up a deposit of ETH that is 1.5× the face-value of what they receive (this is called over-collateralization). An AliceCoin offered for less than \$1 USD can be purchased and redeemed for a profit–assuming the DApp holds enough ETH. Otherwise, an Alice-Coin will sell between \$0 and \$1 USD according to the value of the ETH held by the DApp.

Is it risky for Alice to offer such an AliceCoin? Holding the second place in line is more volatile than holding ETH itself—this stability mechanism does not (and cannot) eliminate volatility, it simply pushes it from the first place to the second place in line. However the second place in line is never more than \$1 USD short of the full amount of ETH held in the DApp. So if Alice keeps the \$1 USD she received for the AliceCoin, it offsets any losses from the second place in line. She has no more risk than holding ETH. The second place in line can also be sold to someone who is seeking risk: the token is a leveraged bet that ETH rises in value. Is it risky for Bob? In most conditions, holding an AliceCoin is purposefully the same as holding USD. However if the USD/ETH rate deteriorates quickly, the AliceCoin will use up its buffer and start to lose value (at the same rate as ETH).

Here are just a few of the design decisions to consider when deploying an indirectly-backed stablecoin: what should the overcollateralization ratio be (*e.g.,* 1.5x)? When can an AliceCoin be redeemed (*e.g.,* on-demand, after an elapsed time, after movements in USD/ETH, *etc.*)? How do you issue multiple AliceCoins (*e.g.,* collateral for each coin is held separately, or collateral for all coins are pooled together and coins are interchangeable)?

6.5 Type 2: Intervention-based Stablecoins

6.5.1 Money Supply Adjustments

A trusted oracle provides the going exchange rate between the cryptocurrency and a stable-valued asset, such as the USD. When the cryptocurrency gains value, the supply of the cryptocurrency is increased, and when it loses value, the supply is decreased. This mechanism is based on how central banks have historically controlled their economies, however the specifics of exchange rate targetting have been abandoned by modern central banks after past failures. That said, exchange rates are an illustrative example and other financial indicators could be used: oracle-provided interest rates (should lending markets emerge) or purchasing power; on-blockchain metrics like transaction volumes (should these prove robust against manipulation), or human discretion (such as central banks themselves [36]).

Allowing a crypto-currency to expand is not difficult. Who receives the new currency is a design decision with options including: (1) existing holders of the currency in proportion to their holdings, (2) existing holders through a random lottery, (3) miners, or (4) a specific entity like a central bank. Who loses when the currency contracts is the primary challenge.

The mechanism in Table 6.2 gives one illustration. Here if many bids for AliceCoin exceed \$1.02 USD, some of the newly injected currency could be spent on obtaining USD until all buyers willing to pay more than \$1.02 USD have purchased AliceCoins. This is merely a heuristical argument because there is no guarantee the recipients will spend the new currency on USD, especially if demand for USD is falling. The justification for offers below \$0.98 is symmetric: the currency contractions could make holders less willing to spend it on USD. However if the price drop is caused by a lack of demand for AliceCoins rather than an oversupply, then removing supply will only thin out the market but not actually incentivize traders to trade and correct the undervaluation.

When the coinbase is increased or decreased dynamically (called an elastic coinbase), increases can be by any amount but decreases cannot appear to go past zero. When the coinbase is exactly zero, miners are still incentivized to mine because of the fees provided in the transactions. In fact this is how Bitcoin will eventually (projected to happen in 2140) function once all BTC is created (how well it will work is debatable [30]). Could the coinbase go negative? Since miners are rewarded the sum of the coinbase and the transaction fees, a coinbase can indeed be moderately negative if the transaction fees are greater than the negative coinbase. Under this deployment, the users are effectively burning their transaction fees to contract the money supply.

6.5.2 Asset Transfer

The second subtype of intervention-based stability mechanism expands and contracts the supply of currency to influence its value, however it uses a less direct contraction method. Recall the mechanism in Table 6.2. If many bids in excess of \$1.02 USD remain unexecuted, the logic follows the previous section: the currency is handed

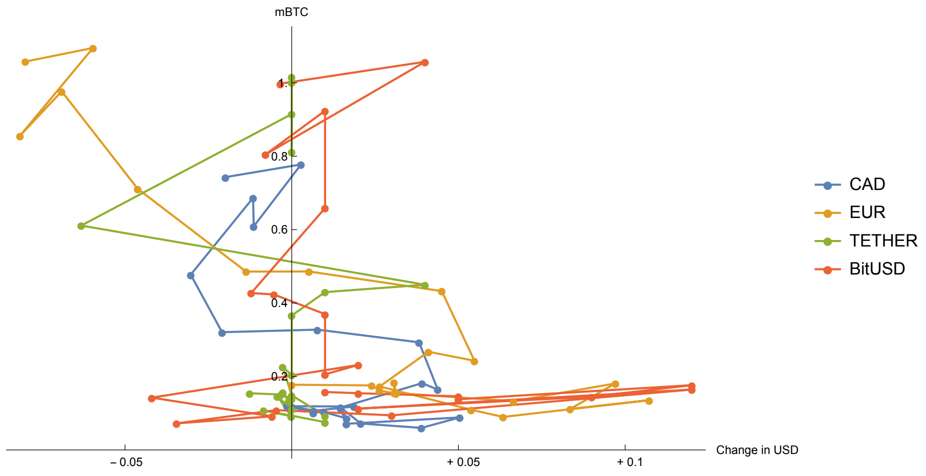


Figure 6.3: Volatility in prices for two fiat currencies (CAD and EUR) and two stablecoins (Tether and BitUSD) against the USD and BTC. Vertical line segments demonstrate stability with the USD. Horizontal shows volatility in USD. While CAD and EUR are not pegged to USD, they demonstrate a degree of stability not that different from the stablecoins. Prices from Jan 2017 to Nov 2018; 1000 mBTC = 1 BTC.

out in hopes that more USD will be bought. The justification for offers below $0.98 is premised on individuals buying places in line, and if this premise is true, the resulting contraction of the currency follows the same logic as the previous section. The purchase of a spot in line is highly speculative — the currency might not return to stability and the spot might never be reached. As the line gets longer, the price of a place in line will fall, and the speculative market will thin out to traders wanting a higher and higher risk/reward ratio. These trends do not guarantee, or even point toward, a recovery in price.

6.6 Discussion Points

Up to this point, we walked through our proposed taxonomy for stabilization techniques and described each mechanism in detail. In this section, we bring up some several important discussions on the stablecoin topic as we believe they make contributions to research on stablecoins. First, we provide an evaluation of stabilization mechanisms based on their trust assumptions and core idea. In the following we discuss oracles, used by some above techniques to obtain the exchange rate. Next, we elaborate on what stability means by visualizing it using a set of graphs. Finally, we explore and discuss the potential stable index-cryptocurrencies (namely Ethereum gas) in the context of stablecoins.

6.6.1 Evaluation of Stabilization Mechanisms

We have considered various issues with different stablecoin mechanisms (Sections 6.4 and 6.5) and summarize our findings in Table 6.3, which evaluates each mechanism's core design and trust model. The table's rows correspond to the mechanisms, and the columns capture evaluation criteria. While all properties contribute to a holistic view of stablecoin design, they do not carry equal weight. Some—such as the ability to correct under- and overvaluation, and the transparency of trust assumptions—are more foundational to achieving stability. Others, like composability or redemption friction, may be secondary. Priorities also vary by context: a user focused on censorship resistance may value decentralization, while a regulator might prioritize collateral verification. Our framework captures these diverse considerations, while acknowledging that their relative importance depends on the use case and stakeholder perspective. This comparative analysis addresses **RQ6(b)** by evaluating how the risk and security profiles of different stablecoin mechanisms compare to one another and to traditional forms of digital cash.

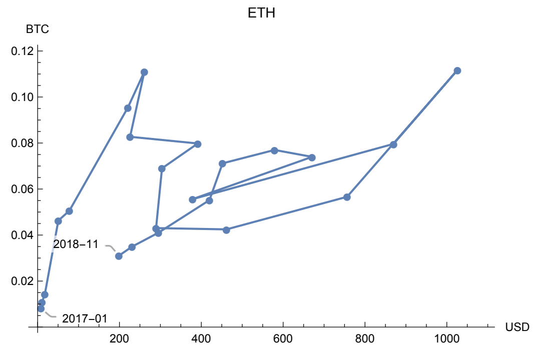
Failures of Intervention-Based Mechanisms: Lessons from Luna and TerraUSD. Intervention-based stablecoins, while promising autonomy and minimal trust reliance, often falter in the absence of robust safeguards. The collapse of the TerraUSD (UST) and Luna ecosystem in May 2022 exemplifies these risks. Designed to maintain a $1 USD peg through a purely algorithmic mechanism, UST's stability relied on arbitrage and mint-and-burn dynamics with Luna. However, this mechanism unraveled when UST lost its peg, triggering a "death spiral" where Luna's oversupply led to catastrophic devaluation. A detailed analysis of this event is provided in [69], published in April 2023.

This failure underscores a broader issue with heuristic and algorithmic stability mechanisms: their reliance on market confidence and idealized assumptions about trader behavior. Without hard reserves or external intervention mechanisms, intervention-based stablecoins remain vulnerable to extreme market stress. The TerraUSD collapse demonstrates that even innovative designs can fail when exposed to the unpredictable dynamics of financial markets, highlighting the need for rigorous stress testing, robust stability models, and regulatory scrutiny. This analysis addresses **RQ6(c)** by evaluating whether the risks associated with intervention-based stablecoins undermine their viability.

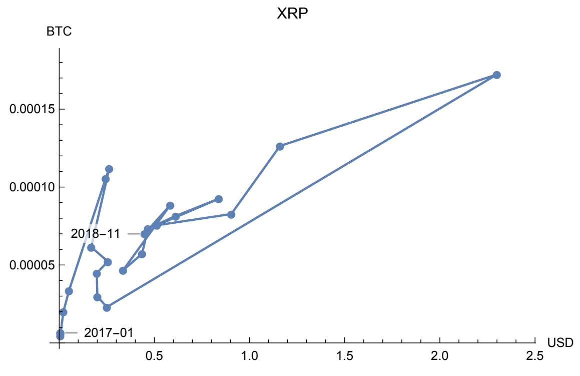
6.6.2 Oracles Everywhere

A number of stablecoin proposals feature oracles which feed information about the stablecoin's exchange rate onto the blockchain. This is essential for indirectly-backed stablecoins, and incidental to intervention-based coins. This raises an important design question. A stablecoin exists so that a digital (or material) good or service can be effectively priced in, say, USD instead of in ETH. To be clear, it is priced in the stablecoin, which maintains a stable exchange rate with USD. To accomplish this, an oracle provides a reliable exchange rate and somewhat elaborates contracts issue stablecoins with an unusual risk profile (*e.g.,* full redemption might not be possible under certain market conditions) that could be difficult for non-experts to understand.

Let us take a step back and think about the larger picture. If we have trustworthy oracles providing reliable exchange rates, is it not a simpler design to just have transacting parties use the oracle directly? Anyone wanting to do business in a stable currency can determine at transaction time how much their good or service, priced in USD, is in ETH and charge the correct amount in ETH (which can then be immediately liquidated for USD, if desired). In summary, the oracle assumption that underlies many stablecoins is itself sufficient to side-step the need for stablecoins. This is applicable to lending as well: loans, interest rates, and repayment amounts can be denominated in USD but paid in ETH using a spot conversion via an oracle.



(a) ETH



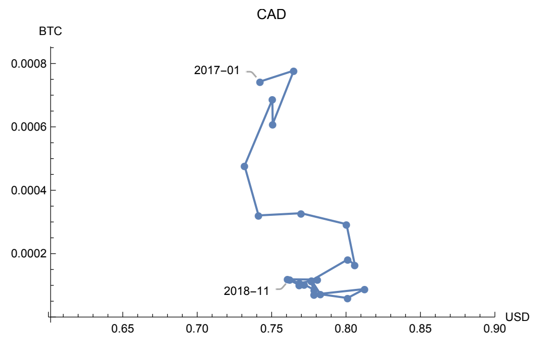
(b) XRP

Figure 6.4: Volatility in cryptocurrencies

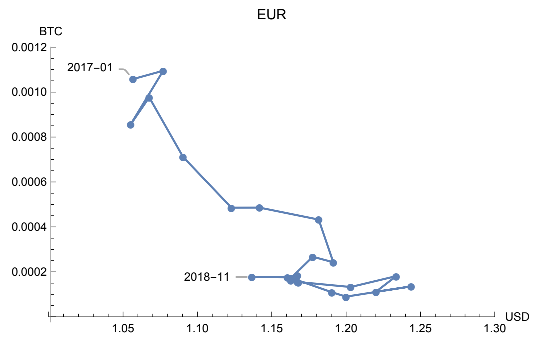
6.6.3 Visualizing Stability

We show a connected scatter plot in Figure 6.4 that shows two cryptocurrencies, ETH and XRP (from Ripple) plotted against two reference currencies: the USD on the x-axis as a currency with government-managed stability, and Bitcoin which has no stability mechanism. Like Bitcoin, neither ETH nor XRP have a stability mechanism. The reader might anticipate one of two things: either (i) they move independently from the reference currencies (diagonal movements along the 2/6 direction) or (ii) they move in a way that is correlated to Bitcoin (3/7 movements) because the market prices all cryptocurrencies like a sector. From Figure 6.4, it is fairly apparent that (i) is correct. The graph displays XRP's strong price surge in December 2017.

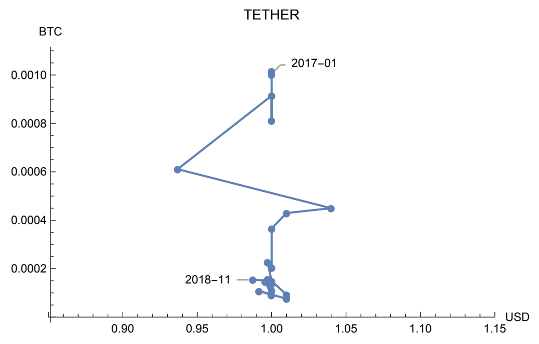
Next we plot a number of stablecoins in Figure 6.5. The top two plots are governmental currencies, the Canadian dollar and the Euro, which have no formal relationship to the USD but are managed by their central banks using similar policies and have intertwined economies. The bottom two currencies are two stablecoins, Tether (directly backed with USD) and BitUSD (indirectly backed with USD). All



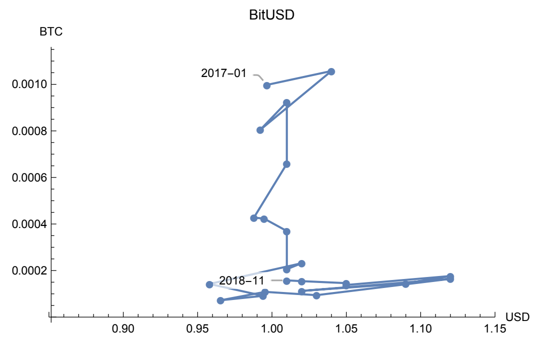
(a) CAD



(b) EUR



(c) Tether

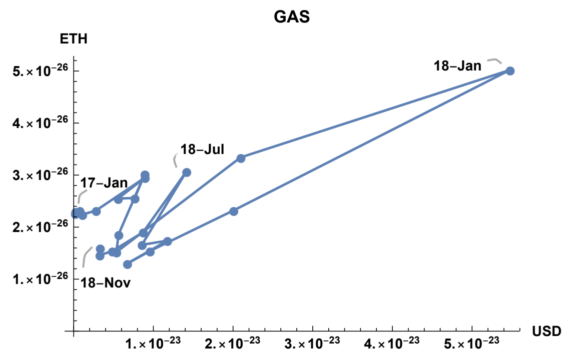


(d) BitUSD

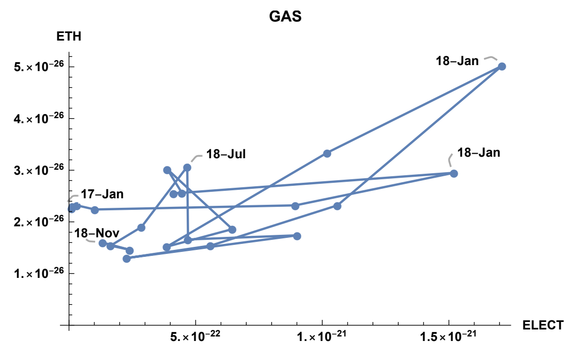
Figure 6.5: Stability in two government-issued fiat currencies (CAD and EUR) and two stablecoin projects (Tether and BitUSD). Note that the x-axis is sized consistently across all four plots, with a $0.30 USD spread.

four currencies exhibit movements in 1/5 direction which indicate that most price movements are due to Bitcoin's volatility and not the volatility of either the plotted currency or USD. Note also that the spread of the x-axis is consistent across all four plots to allow cross-comparison. Both Tether and BitUSD exhibit some volatility. When Tether breaks from its stability with the USD, it moves in diagonal movements that are not correlated with either Bitcoin or USD. When BitUSD loses its stability relative to the USD, it moves in a horizontal 3/7 direction which is correlated with BTC.

To further explore this, we present another visualization in Figure 6.3 that com-



(a) Ethereum gas with respect to ETH and USD.



(b) Ethereum gas with respect to ETH and Electricity.

Figure 6.6: Ethereum average gas price variations with respect to Ether, USD, and Electricity. As mentioned in Section 6.6.4, the drastic movements in the charts represent specific events. Data is from January 2017 to November 2018.

pares two fiat currencies (CAD and EUR) with two stablecoins (Tether and BitUSD), again plotted against USD and BTC. Vertical movements indicate stability with respect to USD, while horizontal movement reflects volatility against USD. Interestingly, although CAD and EUR are not explicitly pegged to the USD, their trajectories show a similar degree of stability to that of the stablecoins. This underscores an important point: even without explicit peg mechanisms, traditional fiat currencies can exhibit price behavior comparable to stablecoins, especially when measured relative to a highly volatile asset like BTC. This visualization reinforces the idea that stability is often contextual—dependent on the reference assets used for comparison—and highlights that not all volatility is created equal.

6.6.4 Ethereum's gas: a stable 'coin'?

DApps on Ethereum execute arbitrary code provided by the owner of the DApp. While this code might be written in a high-level programming language like Solidity, it is compiled to a compact representation (called 'bytecode') that is a set of low-level instructions to the environment (Ethereum virtual machine or EVM). Because different functions will have different complexities, the user running the function pays in proportion to the number of instructions, the complexity of the instructions, and the storage requirements. This means that each operation has a fixed price. Naturally the operations might be priced in ETH, since it is the on-board currency, however this would cause the price of computation to be as volatile as Ether itself. Instead, Ethereum uses a pseudo-currency called gas.[5] Each instruction has a fixed price in gas. A user who wants to run a function will offer to pay a certain amount of ETH per unit of gas to the miner who finalizes the function. Miners will generally choose which functions to run first based on how much ETH/gas they offer, and they might ignore functions that offer too little ETH/gas. We describe gas as a pseudo-currency because it cannot be directly stored or transacted, however we will revisit this below.

Gas was envisioned as maintaining a relatively stable value where a particular function should cost the same amount (say in USD) over time, even as the price of ETH changes dramatically (as seen in Figure 6.4a). We first investigate how successful gas has been with the charts in Figure 6.6, which show the monthly average gas price variations with respect to USD and ETH in the first chart; and electricity and USD in the other. Electricity data is from a Canada-based average index which does not necessarily reflect the costs of mining on a global blockchain, like Ethereum, but if gas were correlated to electricity generally, it should be evident from a representative

[5]http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#what-is-gas

energy index. Gas demonstrates diagonal movements along the 2/6 direction meaning that it actually moves independently of ETH, USD and electricity. There is no strong evidence of stability. This could be due to a few factors. First, the graph is dominated by one large spike and one moderate spike which correspond to (i) when the popular Ethereum game Cryptokitties [6] was first launched (January 2018) and, (ii) when the China-based crypto exchange FCOIN [7] was launched (July 2018) and required a lot of on-chain voting. Both these events clogged up the Ethereum network and increased the gas price as users had to pay more gas for their transactions to go through. Second, it is probably true that users do not have a strong mental model of how much gas to stake for a computation and rely heavily on the user interface for prompts about gas.

Although gas might become a stable unit of account, it is not a store of value because it cannot be held or transacted. However gas could be used to back a stablecoin, much like the coins in the directly-backed category. Amazingly, such a gas-backed coin could even be made redeemable. Ethereum is designed in such a way that it allows users to create a smart contract which stockpiles and swaps gas with other tokens. Operations that store data on Ethereum blockchain modify its global state hence they are very expensive. So in order to incentivize users to free up space on the blockchain, Ethereum refunds the amount of gas users paid if they delete their smart contracts or stored data [118]. GasToken is a directly-backed and redeemable

[6]Cryptokitties website `https://www.cryptokitties.co/`

[7]Fcoin website `https://www.fcoin.com`

tokenization of gas.[8] When the gas price is low (*e.g.,* 1 Gwei), users can store some data on the GasToken contract and create GasTokens. Later when the gas price increases (*e.g.,* 50 Gwei), users can redeem their GasTokens. However they do not receive ETH back; rather, they use the tokens to pay the transaction fees for other computations.

As users' mental model of gas improves over time, the volatility of gas has the potential to reduce. Gas-backed tokens represent a new class of stablecoin that float in value without any direct ties to USD, references to exchange rates, or explicit intervention mechanisms. It is an interesting subject worthy of further research. This addresses **RQ6(d)**, which explores whether gas-backed stablecoins can feasibly provide stability within blockchain ecosystems.

6.7 Concluding Remarks

This chapter has addressed key questions surrounding stablecoins and their mechanisms. We systematically classified stability mechanisms (**RQ6(a)**) and provided concise explanations for each. We evaluated the inherent risks and limitations of intervention-based stablecoins (**RQ6(b)**), highlighting challenges to their viability through examples such as *TerraUSD* in Section 6.5. Additionally, we compared the stability mechanisms to each other and to traditional forms of digital cash, noting trade-offs in trust, decentralization, and stability (**RQ6(b)**). Finally, we explored the feasibility of gas-backed stablecoins (**RQ6(c)**) as a blockchain-native stability

[8]https://github.com/projectchicago/gastoken

solution, and evaluated whether they can serve as a practical unit of account within blockchain ecosystems (**RQ6(d)**).

Stablecoins represent diverse approaches to reducing volatility, with projects differing in redeemable assets, blockchains, and regulatory environments. However, each mechanism involves trade-offs in trust, risk, and feasibility. While directly- and indirectly-backed stablecoins provide tangible anchors to stability, intervention-based designs face significant hurdles in implementation. Looking forward, gas-backed stablecoins offer a compelling avenue for innovation, and CBDCs hold promise for stable and efficient settlements in a digital-first economy. These findings underscore the complexity and importance of stability in decentralized finance.

Chapter 7

Concluding Remarks

This research set out to bridge critical gaps in Decentralized Finance (DeFi), particularly around consumer protection, scalability, and the integration of decentralized systems with traditional financial frameworks. The motivation for this work stemmed from the inherent inefficiencies in traditional finance (TradFi), such as reliance on intermediaries, slow processes, and lack of inclusivity, contrasted with the promise of DeFi's permissionless, transparent, and decentralized nature. By addressing systemic inefficiencies and vulnerabilities, this thesis contributes to aligning the promise of DeFi with practical user needs.

The research explored critical challenges in DeFi, such as front-running, stablecoin risks, decentralized order book scalability, and the inefficiencies of Layer 2 (L2) withdrawal mechanisms. These issues have direct implications for consumer protection, a central theme of this work. By systematically addressing these challenges, this thesis provides actionable solutions, raises new questions, and charts a path for the broader adoption of DeFi.

7.0.1 Key Takeaways and Broader Impact

Each chapter of this thesis addresses a fundamental aspect of DeFi, contributing both theoretical insights and practical solutions:

- **Front-running:** This research proposed a novel taxonomy for front-running attacks and evaluated mitigation techniques to safeguard users in decentralized trading environments. The transparency of blockchain systems creates unique vulnerabilities that require continuous innovation to protect users.

- **Stablecoins:** A systematization of stablecoin mechanisms provided clarity on their operational risks and classifications, enabling better-informed discussions on Central Bank Digital Currencies (CBDCs) and decentralized alternatives. By understanding these systems, regulators can make informed decisions about integrating stablecoins into broader financial ecosystems.

- **Decentralized Order Books:** Through the Lissy framework, this work demonstrated the scalability and performance limitations of on-chain order books, highlighting areas where traditional financial mechanisms may struggle to translate to blockchain environments.

- **Fast Exit Mechanisms:** Tradeable exits for L2 optimistic rollups were developed to mitigate the 7-day withdrawal latency, addressing a critical usability challenge while maintaining security guarantees.

These contributions ensure that the frameworks and tools developed here are not merely theoretical but have practical implications for making DeFi systems more user-friendly, scalable, and aligned with regulatory norms. The broader impact of this research lies in bridging the gap between consumer protection and DeFi innovation, addressing the challenges posed by DeFi's rapid technological advancements and the regulatory frameworks of TradFi.

7.0.2 Revisiting Motivation

This thesis began with a commitment to explore how DeFi could overcome the inefficiencies of TradFi while ensuring consumer protection. The motivation outlined the gaps in accessibility, speed, and cost-efficiency inherent in traditional financial systems, contrasted with DeFi's potential for disintermediation, transparency, and inclusivity.

In addressing these gaps:

- Front-running risks were systematically addressed with practical mitigation strategies, protecting users from exploitative behaviors in decentralized exchanges.

- Stablecoins were dissected to reveal their operational risks and their potential as a cornerstone of decentralized financial systems, particularly in the context of CBDCs and financial stability.

- The scalability of decentralized trading systems, particularly order books, was explored, demonstrating how Layer 2 solutions like Arbitrum can mitigate the high costs of Layer 1 implementations.

- The usability challenges of L2 systems, such as slow withdrawals, were resolved with innovative mechanisms like tradeable exits.

Each contribution ties back to the overarching goal of bridging the divide between DeFi's rapid technological developments and the need for consumer protection.

7.0.3 Research Questions and Contributions in Context

The research questions posed in this thesis guided its focus, addressing systemic issues in DeFi. Subsequent studies have built upon these contributions:

- **Front-running:** Cited works explore cryptographic techniques and decentralized architectures to mitigate front-running risks without sacrificing transparency.
- **Stablecoins:** Further studies examine the integration of CBDCs with DeFi systems and propose frameworks for ensuring stablecoin stability under extreme market conditions.
- **Order Books:** Research has extended this work by exploring hybrid models for order book scalability and front-running mitigation.
- **Fast Exits:** Derivative studies investigate more efficient tradeable exit mechanisms and their applications in cross-chain liquidity solutions.

These citations confirm the relevance and applicability of this research, highlighting its foundational role in DeFi innovation.

7.0.4 Future Work

While significant progress has been made, the journey is far from complete. Future research could explore:

- **Layer 2 Pain Points:** Issues like cross-chain interoperability, data availability, and fraud-proof optimization remain critical challenges.
- **Stablecoins and CBDCs:** Understanding the role of CBDCs in DeFi and addressing operational risks in decentralized stablecoins are promising directions.
- **Front-running Mitigation:** Advanced cryptographic techniques and fair-ordering mechanisms could provide stronger protections against exploitative behaviors.
- **TradFi Mechanisms in DeFi:** Investigating which traditional financial mechanisms (e.g., insurance payouts, real-time auctions for swaps, and clearing systems) can be adapted for blockchain environments.

Addressing these challenges will further align DeFi with its potential to create an accessible, efficient, and secure financial ecosystem.

By bridging the divide between consumer protection and decentralized innovation, this research has laid the groundwork for a future where DeFi is both inclusive and robust, empowering users globally.

Bibliography

[1] Ethereum development tutorial · ethereum/wiki wiki. `https://github.com/ethereum/wiki/wiki/Ethereum-Development-Tutorial`. Accessed on 24/08/2020.

[2] 96th cong, 1st sess, report of the special study of the options markets to the securities and exchange comission, 1978.

[3] Im-2110-3. front running policy. Financial Industry Regulatory Authority, 2002.

[4] Ssac advisory on domain name front running. ICANN Advisory Committee, 10 2007. (Accessed on 08/15/2018).

[5] 5270. front running of block transactions. Financial Industry Regulatory Authority, 2012.

[6] Notice of filing of proposed rule change to adopt finra rule 5270 (front running of block transactions) in the consolidated finra rulebook. SECURITIES AND EXCHANGE COMMISSION, 2012.

[7] A brief history of Stablecoins (Part 1) – BitMEX Blog, Aug 2019. [Online; accessed 21. Aug. 2019].

[8] Stablecoins: designing a price-stable cryptocurrency, Aug 2019. [Online; accessed 21. Aug. 2019].

[9] Stablecoins vs. Govtcoins: The Race to Solve Cryptocurrencies' Price Volatility Problems, Aug 2019. [Online; accessed 21. Aug. 2019].

[10] F. M. Ametrano. Hayek money: The cryptocurrency price stability solution. *Available at SSRN 2425270*, 2016.

[11] M. Aquilina, E. B. Budish, and P. O'Neill. Quantifying the high-frequency trading "arms race": A simple new methodology and estimates. *Chicago Booth Research Paper*, (20-16), 2020.

[12] Aragon. Build better, together - aragon dao.

[13] R. T. Aune, A. Krellenstein, M. O'Hara, and O. Slama. Footprints on a blockchain: Trading and information leakage in distributed ledgers. *The Journal of Trading*, 12(3):5–13, 2017.

[14] T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten. Have a snack, pay with bitcoins. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–5. IEEE, 2013.

[15] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 307–323. Springer, 1992.

[16] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable zero knowledge with no trusted setup. In *CRYPTO*, 2019.

[17] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza. Snarks for c: Verifying program executions succinctly and in zero knowledge. In *CRYPTO*, 2013.

[18] S. Blockchain Luxembourg. The state of stablecoins, 2018.

[19] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO*, 2018.

[20] J. Bonneau, J. Clark, and S. Goldfeder. On bitcoin as a public randomness source. `https://eprint.iacr.org/2015/1015.pdf`, 2015. Accessed: 2015-10-25.

[21] J. Bonneau, E. W. Felten, A. Miller, and A. Narayanan. Decentralizing blockchain security: Proposer-builder separation and beyond. *Cryptography and Security*, 2015:1–20, 2015.

[22] R. Brandom. This princeton professor is building a bitcoin-inspired prediction market, Nov 2013.

[23] L. Breidenbach, I. Cornell Tech, P. Daian, F. Tramer, and A. Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. {USENIX} Association, 2018.

[24] E. Budish, P. Cramton, and J. Shim. The high-frequency trading arms race: Frequent batch auctions as a market design response. *The Quarterly Journal of Economics*, 130(4):1547–1621, 2015.

[25] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In

USENIX Security Symposium, Baltimore, MD, Aug. 2018. USENIX Association.

[26] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 319–338.

[27] B. Bünz, S. Goldfeder, and J. Bonneau. Proofs-of-delay and randomness beacons in ethereum. In *IEEE S&B*, 2017.

[28] V. Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3:37, 2014.

[29] S. Buti, B. Rindi, and I. M. Werner. Diving into dark pools. 2011.

[30] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 acm sigsac conference on computer and communications security*, pages 154–167, 2016.

[31] J. Cartlidge, N. P. Smart, and Y. Talibi Alaoui. Mpc joins the dark side. In *ASIACCS*, pages 148–159, 2019.

[32] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE EuroS&P*, pages 185–200. IEEE, 2019.

[33] J. Clark, J. Bonneau, E. W. Felten, J. A. Kroll, A. Miller, and A. Narayanan. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security (WEIS)*, volume 188, 2014.

[34] J. Clark, H. Cummins, A. Diamanti, A. Kurmus, and F. Tschorsch. Sok: Towards censorship-resistant transaction relay, 2018.

[35] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. In *IEEE Symposium on Security and Privacy*, 2020.

[36] G. Danezis and S. Meiklejohn. Centrally banked cryptocurrencies. *arXiv preprint arXiv:1505.06895*, 2015.

[37] D. Demirag and J. Clark. Absentia: Secure multiparty computation on ethereum. In *Workshop on Trusted Smart Contracts (WTSC)*, pages 381–396. Springer, 2021.

[38] W. Diffie and M. Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[39] B. Edelman. Front-running study: Testing report, 2009.

[40] S. Eskandari, S. Moosavi, and J. Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *WTSC*. FC, 2019.

[41] S. Eskandari, S. Moosavi, and J. Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security*, pages 170–189. Springer, 2019.

[42] Ethereum Foundation. The Search for a Stable Cryptocurrency, Aug 2019. [Online; accessed 21. Aug. 2019].

[43] E. Félez-Viñas and B. Hagströmer. Do volatility extensions improve the quality of closing call auctions? *Financial Review*, 56(3):385–406, 2021.

[44] FinancialTimes. Barclays trader charged with front-running by us authorities. 2018.

[45] B. Ford and R. Böhme. Rationality is self-defeating in permissionless systems. Technical Report cs.CR 1910.08820, arXiv, 2019.

[46] B. Ford and R. Böhme. Rationality is self-defeating in permissionless systems. *arXiv preprint arXiv:1910.08820*, 2019.

[47] N. Foundation. Hardhat. `https://hardhat.org`, October 2022. (Accessed on 10/18/2022).

[48] H. S. Galal and A. M. Youssef. Publicly verifiable and secrecy preserving periodic auctions. In *WTSC*. Springer, 2021.

[49] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.

[50] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *EUROCRYPT*, 2013.

[51] J. M. Griffin and A. Shams. Is bitcoin really un-tethered? *SSRN Electronic Journal*, October 28 2019. Available at SSRN: `https://ssrn.com/abstract=3195066` or `http://dx.doi.org/10.2139/ssrn.3195066`.

[52] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography*, pages 201–226. Springer, 2020.

[53] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais. Sok: Layer-two blockchain protocols. In *Financial Cryptography*, 2020.

[54] A. Hafid, A. S. Hafid, and M. Samih. Scaling blockchains: A comprehensive survey. *IEEE access*, 8:125244–125262, 2020.

[55] L. Harris. *Trading and exchanges: market microstructure for practitioners.* Oxford, 2003.

[56] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoins peer-to-peer network. In *USENIX Security*, pages 129–144, Washington, D.C., 2015. USENIX Association.

[57] P. Hillion and M. Suominen. The manipulation of closing prices. *Journal of Financial Markets*, 7(4):351–375, 2004.

[58] J. Hull, S. Treepongkaruna, D. Colwell, R. Heaney, and D. Pitt. *Fundamentals of futures and options markets.* Pearson Higher Education AU, 2013.

[59] initc3.org. Frontrun me. *URL: http://frontrun.me/*, 2018.

[60] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security Symposium*, pages 1353–1370, 2018.

[61] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan. An empirical study of namecoin and lessons for decentralized namespace design. In *WEIS*. Citeseer, 2015.

[62] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.

[63] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.

[64] R. Khalil, A. Gervais, and G. Felley. Tex-a securely scalable trustless exchange. *IACR Cryptol. ePrint Arch.*, 2019:265, 2019.

[65] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *IEEE Symposium on Security and Privacy*, pages 1–19, 2019.

[66] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.

[67] K. Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, pages 25–36, 2020.

[68] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg. Meltdown: Reading kernel memory from user space. In *USENIX Security Symposium*, pages 973–990, Baltimore, MD, Aug. 2018. USENIX Association.

[69] J. Liu, I. Makarov, and A. Schoar. Anatomy of a run: The terra luna crash. Technical report, National Bureau of Economic Research, 2023.

[70] Y. Marcus, E. Heilman, and S. Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236, 2018. `https://eprint.iacr.org/2018/236`.

[71] J. W. Markham. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.*, 38:69, 1988.

[72] F. Massacci, C. N. Ngo, J. Nie, D. Venturi, and J. Williams. Futuresmex: secure, distributed futures market exchange. In *IEEE Symposium on Security and Privacy*, pages 335–353. IEEE, 2018.

[73] V. Mavroudis and H. Melton. Libra: Fair order-matching for electronic financial exchanges. In *ACM AFT*, 2019.

[74] G. Maxwell. Confidential transactions. *URL: https://people. xiph. org/˜ greg/-confidential_values. txt (Accessed 09/05/2016)*, 2015.

[75] P. McCorry, C. Buckland, B. Yee, and D. Song. Sok: Validating bridges as a scaling solution for blockchains. Technical report, Cryptology ePrint Archive, 2021.

[76] P. McCorry, S. F. Shahandashti, and F. Hao. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*, pages 357–375. Springer, 2017.

[77] S. Meiklejohn. An evolution of models for zero-knowledge proofs. In *EUROCRYPT (invited talk)*, 2021.

[78] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[79] A. Moin, K. Sekniqi, and E. G. Sirer. Sok: A classification framework for stablecoin designs. In *Financial Cryptography*, 2020.

[80] M. Moosavi and J. Clark. Lissy: Experimenting with on-chain order books. In *Workshop on Trusted Smart Contracts (WTSC)*, 2021.

[81] A. Müller and M. Grandi. Weather derivatives: a risk management tool for weather-sensitive industries. *The Geneva Papers on Risk and Insurance. Issues and Practice*, 25(2):273–287, 2000.

[82] T. Nadahalli, M. Khabbazian, and R. Wattenhofer. Grief-free atomic swaps. In *2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 1–9. IEEE, 2022.

[83] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[84] S. Nakamoto. *The book of satoshi: the collected writings of Bitcoin creator satoshi nakamoto*. Phil Champagne, 2014.

[85] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[86] A. Narayanan, J. Bonneau, E. W. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies*. Princeton, 2016.

[87] S. Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. `https://eprint.iacr.org/2015/1098`.

[88] A. Norry. The history of the mt gox hack: Bitcoin's biggest heist. `https://blockonomi.com/mt-gox-hack/`, June 2019. (Accessed on 12/31/2019).

[89] C. O'Higgins. Stablecoins - everything you need to know. `https://cryptoinsider.21mil.com/stablecoins-everything-need-know/`, 2018. Accessed: 2018-07-09.

[90] Optimism. Optimism is ethereum, scaled. online, 2022.

[91] M. S. Pagano and R. A. Schwartz. A closing call's impact on market quality at euronext paris. *Journal of Financial Economics*, 68(3):439–484, 2003.

[92] A. Park. The conceptual flaws of constant product automated market making. *Available at SSRN 3805750*, 2021.

[93] I. G. A. Pernice, S. Henningsen, R. Proskalovich, M. Florian, and H. Elendner. Monetary stabilization in cryptocurrencies: Design approaches and open questions. In *CVCBT*, 2019.

[94] C. P. Pfleeger and S. L. Pfleeger. *Security in computing*. Prentice Hall Professional Technical Reference, 2002.

[95] E. Pimentel, E. Boulianne, S. Eskandari, and J. Clark. Systemizing the challenges of auditing blockchain-based assets. *Journal of Information Systems*, 35(2):61–75, 2021.

[96] R. Radner and A. Schotter. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory*, 48(1):179–220, 1989.

[97] H. Ragab, A. Milburn, K. Razavi, H. Bos, and C. Giuffrida. Crosstalk: Speculative data leaks across cores are real. In *IEEE Symposium on Security and Privacy*, 2021.

[98] S. Ruoti, B. Kaiser, A. Yerukhimovich, J. Clark, and R. Cunningham. Blockchain technology: What is it good for? *Communications of the ACM*, 63(1), 2020.

[99] R. Sams. A note on cryptocurrency stabilisation: Seigniorage shares. [Online], 2015.

[100] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.

[101] B. Schneier. *Applied cryptography: protocols, algorithms, and source code in C.* john wiley & sons, 2007.

[102] Securities and E. B. of India. Sebi — order in the matter of nse colocation. `https://www.sebi.gov.in/enforcement/orders/apr-2019/order-in-the-matter-of-nse-colocation_42880.html`, 2019. (Accessed on 11/11/2019).

[103] N. Sexer. State of Stablecoins, 2018. *Medium*, Aug 2019.

[104] A. Shevchenko. Uniswap and automated market makers, explained. *URl: https://cointelegraph.com/explained/uniswap-and-automated-market-makers-explained*, 2020.

[105] C. Signer. Gas cost analysis for ethereum smart contracts. Master's thesis, ETH Zurich, Department of Computer Science, 2018.

[106] T. Suite. Ganache. `https://www.trufflesuite.com/ganache`, May 2021. (Accessed on 05/26/2021).

[107] T. Suite. Truffle. `https://www.trufflesuite.com/docs/truffle/overview`, May 2021. (Accessed on 05/26/2021).

[108] N. Szabo. Formalizing and securing relationships on public networks. *First monday*, 1997.

[109] P. Sztorc. Truthcoin. Technical report, 2015.

[110] B. Team. Introducing: 2019 State of Stablecoins Report. *Blockchain Blog*, Feb 2019.

[111] C. P. Team. Comprehensive overview of STABLECOINS. *Medium*, Oct 2018.

[112] L. T. Thibault, T. Sarry, and A. S. Hafid. Blockchain scaling using rollups: A comprehensive survey. *IEEE Access*, 10:93039–93054, 2022.

[113] C. Thorpe and D. C. Parkes. Cryptographic securities exchanges. In *Financial Cryptography*, 2007.

[114] C. Thorpe and S. R. Willis. Cryptographic rule-based trading. In *Financial Cryptography*, 2012.

[115] R. Ver and J. Wu. Bitcoin cash planned network upgrade is complete. 2018. Accessed: 2018-12-07.

[116] J. Vermorel, A. Séchet, S. Chancellor, and T. van der Wansem. Canonical transaction ordering for bitcoin. 2018. Accessed: 2018-12-07.

[117] D. Z. J. Williamson. The aztec protocol. *URL: https://github.com/AztecProtocol/AZTEC/*, 2018.

[118] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. Technical Report 2014, Ethereum, 2014.

[119] W. Yuen, P. Syverson, Z. Liu, and C. Thorpe. Intention-disguised algorithmic trading. In *Financial Cryptography*, 2010.

[120] A. Zamyatin, M. Al-Bassam, D. Zindros, E. Kokoris-Kogias, P. Moreno-Sanchez, A. Kiayias, and W. J. Knottenbelt. Sok: Communication across distributed ledgers. In *Financial Cryptography*, 2021.

[121] L. Zhao, J. I. Choi, D. Demirag, K. R. B. Butler, M. Mannan, E. Ayday, and J. Clark. One-time programs made practical. In *Financial Cryptography*, 2019.

[122] H. Zhu. Do dark pools harm price discovery? *The Review of Financial Studies*, 27(3):747–789, 2014.