RETHINKING SECURITY FOR EMERGING DECENTRALIZED SYSTEMS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Mahimna Kelkar

August 2025

© 2025 Mahimna Kelkar

ALL RIGHTS RESERVED

RETHINKING SECURITY FOR EMERGING DECENTRALIZED SYSTEMS

Mahimna Kelkar, Ph.D.

Cornell University 2025

In the past decade, blockchains have emerged as a new class of decentralized systems and are regularly viewed as disruptive technology with many applications. For security research, blockchains are of particular interest because they operate in a radically new, highly adversarial environment where protocol flaws and software bugs can be immediately monetized by anonymous actors.

This dissertation focuses on the complexity of designing protocols that are secure in this new adversarial environment. In the process, it questions established modeling assumptions in well-studied, longstanding security frameworks, and finds novel insights about classical security notions. It aims to highlight how the blockchain environment provides a guiding principle for how we should think about the robustness of our systems more broadly.

The first part of this dissertation shows how a prominent blockchain application, decentralized finance, surfaces an under-specification in the well-studied consensus primitive from distributed systems research.

The second part shows how new capabilities like trusted hardware, multi-party computation, and smart contracts, which are typically used for building secure systems, can also be employed by attackers for nefarious purposes. This results in powerful new attacks that surface subtle gaps in existing literature on proofs of knowledge and accountability in cryptography.

The work in this dissertation benefits from techniques rooted in several areas—distributed systems, cryptography, mechanism design and game theory,

and social choice. The hope is that this further motivates an intersectional lens for blockchain (and more generally, modern systems) security.

BIOGRAPHICAL SKETCH

Mahimna Kelkar completed his undergraduate at Purdue University, where he majored in Computer Science, Math, and Statistics. At Purdue, Aniket Kate introduced him to security research and blockchains in particular, which fascinated him enough to pursue a PhD in the area.

In 2018, Mahimna began his PhD at Cornell, where he was fortunate to be advised by Ari Juels. He spent the first year in gorgeous Ithaca before moving to the Cornell Tech campus based in New York City for the remainder.

During his PhD, Mahimna worked broadly on cryptography, distributed systems, and blockchain security. He completed internships and research visits at Google, Facebook Novi, the Technion, and a16z crypto. He also worked closely with Yuval Ishai (Technion) and Edward Felten (Offchain Labs).

After his PhD, Mahimna will spend a year as a postdoc at Columbia University (hosted by Tim Roughgarden) before starting as an assistant professor of computer science at Purdue in Fall 2026.

This dissertation is dedicated to my parents, who despite their successful careers and busy schedules, somehow always have time for me—even for the most minor of things.

ACKNOWLEDGEMENTS

It is often joked that the acknowledgements section of a PhD dissertation is the hardest one to write and the only part that everyone reads. I didn't quite appreciate the truth behind this until I started to write my own one. Indeed, while earning a PhD is ultimately about becoming *the* world-expert in a deep and novel research question, the journey itself is defined more by the people who support and shape it than by the work alone. Not unlike the works of others before me, this dissertation would not have been possible without many, many people.

Foremost, I am extremely grateful to my PhD advisor, Ari Juels. Throughout my PhD, Ari has been a phenomenal mentor and taught me a great deal, both professionally and personally. He has always been patient and offered sound advice for any situation. Ari has been instrumental in not only guiding the research in this dissertation but also in shaping my overall research philosophy and my approach to choosing impactful problems to work on. His remarkable advising style is something I hope to learn from as I make my own transition to a faculty role.

I appreciate Tom Ristenpart and Rafael Pass for serving on my committee. Over the years, Tom is someone I have sought out for research discussions and career guidance, which he has provided in spades. Rafael in particular, was very helpful for discussions on the Complete Knowledge work (Chapter 9 of this dissertation).

My research has been highly collaborative with over 80 co-authors, which has allowed me to explore a broad range of topics and learn from other researchers. I count myself undeservingly lucky for this.

During my PhD, I was graciously hosted during several industry

internships—by Mariana Raykova and Karn Seth at Google, by Mathieu Baudet and Kostas Chalkias at Facebook Novi, and by Joe Bonneau, Valeria Nikolaenko, and Tim Roughgarden at a16z crypto. Discussions during my stay at a16z crypto were a significant inspiration for the research vision in this dissertation.

I am also fortunate to have had a long and fruitful collaboration with Yuval Ishai at the Technion, who apart from being an exceptional researcher and a generational talent, is also one of the nicest and most caring people I have ever met. Yuval taught me a lot about how to think like a theoretician. He is the reason I started and will continue working on research outside of blockchains.

On the flip side, Ed Felten at Offchain Labs brought critically important practical perspectives on my research through a unique combination of essentially three careers—professor emeritus, technology advisor to the US government, and currently startup co-founder. It has been an absolute pleasure to collaborate with Ed and others at Offchain including Akaki and Ben to deploy some of our research on the Arbitrum blockchain.

The work in this dissertation also benefited through collaboration and insightful discussions with many other senior researchers—most notably, Ittay Eyal, Steven Goldfeder, Aniket Kate, Sreeram Kannan, Dahlia Malkhi, and Matt Weinberg. Aniket introduced me to cryptography and blockchains when I was an undergrad and I am forever thankful for this.

I am happy to have met a wonderful cohort of students during my PhD. My first year, spent in gorgeous but secluded Ithaca, was made enjoyable by many others—notably, Ethan, Jesse, Kate, Kwanghyun, Rachit, Raunak, and Soham. At Cornell Tech in NYC, the security and cryptography researchers formed a tight-knit group that fostered a nurturing environment both academically and socially. There are too many to name here but I appreciate all of you for that.

Special shout-out to Carolina, Deepak, Kushal, Phil, Shachi, Yueying, and Yan at Cornell Tech. I also feel lucky to have collaborated with many friends external to Cornell: Aadityan, Amit, Nusret, Soubhik, and Yiping.

Friends and extended family in the tri-state area offered support, a warm and welcoming home, and were always a pleasure to be around. Thanks especially to the Agashes and Fayaz. My cousin Parth deserves special appreciation for being exceptionally selfless and always looking out for me.

Finally, my most heartfelt appreciation is for my parents and their unwavering support. Nothing could put into words what you have given me and no amount of praise would be sufficient.

TABLE OF CONTENTS

Biographical Sketch . . . iii
Dedication . . . iv
Acknowledgements . . . v
Table of Contents . . . viii
List of Tables . . . xiv
List of Figures . . . xv

1 Introduction **1**
1.1 Origins . . . 1
1.2 Blockchain Research . . . 2
1.3 This Dissertation . . . 3
1.3.1 Decentralized Finance and Transaction Ordering . . . 4
1.3.2 Insights from New Attacker Capabilities . . . 5

I Insights from New Applications: Blockchain Transaction Ordering **8**

2 Order-Fair Consensus **11**
2.1 Order-manipulation Attacks and the Importance of Fair Transaction Ordering . . . 12
2.2 Preliminaries and Model . . . 15
2.2.1 Protocol Execution Model . . . 15
2.2.2 Execution Environments . . . 20
2.2.3 The State Machine Replication Abstraction . . . 23
2.3 Defining Fair Ordering . . . 27
2.3.1 Send-Order-Fairness . . . 28
2.3.2 Receive-Order-Fairness . . . 29
2.3.3 Condorcet paradox and the impossibility of fair ordering . 30
2.3.4 Environments that support receive-order-fairness . . . 32
2.3.5 Towards weaker definitions for order-fairness . . . 32
2.4 Comparing Order-Fairness to Related Works . . . 35
2.5 Other Order-Fairness Notions . . . 40
2.5.1 Hashgraph Details . . . 41

3 The Aequitas Protocol Family **43**
3.1 Building Blocks . . . 43
3.1.1 Set Byzantine Agreement . . . 44
3.1.2 FIFO Broadcast . . . 48
3.2 General Design Overview . . . 50
3.2.1 The Finalization Stage . . . 57
3.2.2 Adding Order-Fairness to Any Consensus Protocol . . . 60

3.3 The Synchronous Aequitas protocol . . . 61
3.3.1 Protocol Description . . . 61
3.3.2 Protocol Pseudocode . . . 65
3.3.3 Consistency . . . 68
3.3.4 Liveness . . . 69
3.3.5 Batch-order-Fairness . . . 70
3.3.6 Modified protocol for $\delta_{\text{ext}} \leq 1$ . . . 70
3.4 The Asynchronous Aequitas protocol . . . 71
3.4.1 Protocol Pseudocode . . . 72
3.4.2 Asynchronous Aequitas Properties . . . 74
3.5 Leader-Based Aequitas Protocols . . . 75

4 A New Social Choice Theory Perspective on Order-Fairness 77
4.1 The Finalization Problem . . . 77
4.2 Static-Finalization as Social Choice Aggregation . . . 79
4.3 A New Streaming Social Choice Variant . . . 80
4.3.1 Aequitas Solves Streaming Finalization . . . 83

5 Order-Fairness in the Permissionless Setting 85
5.1 Model and Preliminaries . . . 85
5.1.1 Communication Networks . . . 87
5.1.2 Remark on the Network Model . . . 89
5.1.3 Permissionless Formalism . . . 91
5.1.4 Order-Fairness . . . 93
5.2 Protocol Overview . . . 95
5.2.1 Player Replaceability . . . 97
5.3 Permissionless Protocols . . . 100
5.3.1 Modulo-T Longest Chain Protocol . . . 100
5.3.2 Multi-Chain Protocol . . . 103
5.3.3 Fruitchains version of Π_{mod} . . . 105
5.4 Proof Sketches . . . 108
5.4.1 Overview of Proof Technique . . . 108
5.4.2 Proof Sketch for Π_{mod} . . . 110
5.4.3 Proofs Sketch for Π_{multi} . . . 111
5.4.4 Proof Sketch for Π_{fruit} . . . 114
5.5 Applications . . . 114
5.5.1 Zero-Block Confirmation . . . 115
5.5.2 Soft-Ordering as Hamiltonian paths . . . 117

6 Themis: Fast, Strong Order-Fairness 120
6.1 Themis Overview and Contributions . . . 120
6.2 Themis Model and Preliminaries . . . 125
6.3 Building Blocks . . . 127
6.3.1 Aequitas Background . . . 127

6.3.2 Aequitas Technical Challenges . . . 129
6.3.3 Novel Understanding of Condorcet Cycles . . . 131
6.4 Themis Description . . . 136
6.4.1 Constructing the Leader Proposal . . . 137
6.4.2 Finalizing the Fair Ordering . . . 144
6.4.3 Formal Themis Results . . . 147
6.4.4 SNARK-Themis: A SNARK-based variant . . . 148
6.5 Implementation and Benchmarks . . . 151
6.5.1 Performance and Benchmarks . . . 152

7 Suite of Fair Ordering Experiments 155
7.1 Fair-Ordering Definitions . . . 155
7.2 Simulation Environment . . . 157
7.3 Ideal Setting Comparison . . . 159
7.4 Network Level Insertions and Frontrunning . . . 162
7.4.1 Formal analysis in a natural network setting. . . . 163
7.4.2 Experimental analysis. . . . 165
7.5 Robustness to Adversarial Reordering . . . 167
7.6 Censorship Resistance and Attacks . . . 171

8 TimeBoost: Latency Racing vs Bidding for Transaction Ordering 176
8.1 Introduction . . . 176
8.1.1 Existing Ordering Policies . . . 178
8.1.2 Our contributions . . . 182
8.2 Ordering Policies . . . 184
8.2.1 Preliminaries . . . 184
8.2.2 Independence of Irrelevant Transactions (IIT) . . . 185
8.2.3 IIT Implies a Score-Based Policy . . . 186
8.3 TimeBoost Description . . . 190
8.3.1 TimeBoost Economic Analysis Overview . . . 192
8.4 Analysis of TimeBoost with 2 Players . . . 193
8.4.1 Ex-Ante Latency Investment . . . 194
8.4.2 Ex-Post Latency with Bidding . . . 202
8.5 Analysis of TimeBoost with n players . . . 209
8.6 Comparison of TimeBoost with a Pure Bidding Policy . . . 212
8.7 Discussion on Sequencer Decentralization . . . 214
8.8 Conclusion . . . 215

II Insights from New Attacker Capabilities 216

9 Proofs of Complete Knowledge 217
9.1 Introduction . . . 217
9.2 Motivation: Attacks & Applications . . . 224

9.2.1 Attacks and CK-Based Countermeasures . . . 224
9.2.2 New Applications Enabled by CK . . . 227
9.3 Preliminaries and Background . . . 230
9.3.1 Interactive Proof Systems . . . 230
9.4 Proofs of Complete Knowledge . . . 236
9.4.1 Building Intuition . . . 237
9.4.2 Formal PoCK Security . . . 240
9.4.3 Zero-Knowledge PoCK . . . 244
9.4.4 Eavesdropper Undetectability . . . 245
9.5 SGX-based PoCK Protocol . . . 247
9.5.1 SGX-PoCK Properties . . . 249
9.6 ASIC-Based PoCK Construction . . . 252
9.6.1 Background: Mining ASICs and PoWs . . . 253
9.6.2 ASIC-Based PoCK: Protocol construction . . . 255
9.6.3 Security Analysis . . . 259
9.6.4 Practical ASIC-ZKPoCK Parameterization . . . 263
9.7 SMACK: End-to-End Implementation . . . 265
9.7.1 ASIC-ZKPoCK implementation . . . 266
9.7.2 Lightweight CK . . . 268
9.8 Related Work . . . 273
9.9 Conclusion . . . 275

10 Smart Collusion **276**
10.1 Introduction . . . 276
10.1.1 Our Contributions and Results . . . 279
10.1.2 Additional Related Works . . . 284
10.2 Preliminaries . . . 286
10.2.1 Cryptographic Preliminaries . . . 286
10.2.2 Mechanism Design Preliminaries . . . 287
10.3 Whistleblowing Framework . . . 288
10.4 Transparent Service Model . . . 291
10.4.1 A Generic Computation Functionality . . . 292
10.4.2 Whistleblowing Formalism . . . 295
10.4.3 Results on Whistleblowing Feasibility . . . 297
10.4.4 Applications . . . 301
10.5 General n-party Model . . . 304
10.5.1 Applications . . . 306
10.6 Game-Theoretic Model . . . 308
10.6.1 Base game $\mathbb{B}$ . . . 310
10.6.2 Whistleblowing game $\mathbb{W}$ . . . 310
10.6.3 Retaliation Game $\mathbb{R}$ . . . 318
10.7 Effective Whistleblowing . . . 323
10.7.1 The 1-Whistle Mechanism $\mathbb{M}^{\mathbb{W}}_{1*}$ . . . 323
10.7.2 Lower Bounds . . . 324

10.7.3 Alternative Collusion Models . . . 325
10.8 Beyond (n, n)-Collusion . . . 326
10.8.1 (t, n)-Collusion . . . 326
10.8.2 Transparent Service Setting . . . 327

11 Closing Remarks **330**
11.1 How Blockchains Can Inform Modern Systems Security . . . 330

III Appendix **332**

A Appendix for Chapter 2 **333**
A.1 Deferred Proofs for Fairness Properties . . . 333

B Appendix for Chapter 3 **335**
B.1 Deferred Proofs for Aequitas Building Blocks . . . 335
B.2 Deferred Proofs for the Synchronous Aequitas Protocol . . . 336
B.2.1 Consistency Proof . . . 336
B.2.2 Liveness Proof . . . 338
B.2.3 Fairness Proof . . . 339
B.2.4 Modified Synchronous Protocol . . . 339
B.3 Deferred Proofs for the Asynchronous Aequitas Protocol . . . 341
B.3.1 Consistency Proof . . . 341
B.3.2 Liveness Proof . . . 343
B.3.3 Fairness Proof . . . 344

C Appendix for Chapter 5 **345**
C.1 Standard Permissionless Blockchain Formalism . . . 345
C.2 Player Replaceability Proof . . . 347
C.3 Protocol Security Proofs . . . 348
C.3.1 Security Proof for Π_{mod} (Theorem 5.6) . . . 349
C.3.2 Security Proof for Π_{multi} (Theorem 5.7) . . . 353
C.4 Cross-chain-quality proof for Π_{multi} . . . 355
C.4.1 Proof structure. . . . 357
C.4.2 Honest block under $V_i[r]$ . . . 359
C.4.3 Computation of $\phi(f, \beta)$ . . . 362
C.4.4 Achieving cross-chain-quality . . . 374
C.5 Soft-Ordering Application Proofs . . . 376

D Appendix for Chapter 6 **379**
D.1 Security Proofs for Themis . . . 379
D.1.1 Helper Lemmas . . . 379
D.1.2 Proof of Themis Fairness (Theorem 6.4) . . . 381
D.1.3 Proof of Themis Consistency and Liveness (Theorem 6.5) . 383
D.2 Themis Experiment Details . . . 383

D.3 Additional Insights on Related Works 385

E Appendix for Chapter 9 **388**
E.1 $\mathcal{G}_{\mathsf{SGX}}$ Functionality 388
E.1.1 Security Proof for SGX-PoCK 388
E.2 Deferred Details for ASIC-ZKPoCK 392
E.2.1 Proofs 392
E.2.2 Parameterization Details 394
E.3 Deferred Details on SMACK 396

F Appendix for Chapter 10 **399**
F.1 Deferred Cryptography Details 399
F.2 Deferred Cryptography Proofs 402
F.2.1 Impossibility of Whistleblowing for Oblivious Leakage (Lemma 10.11) 402
F.2.2 Whistleblowing Protocol for other Leakages 404
F.3 Existence of a "Reasonable" Second-Stage Pure Nash Equilibrium 406
F.4 Alternate Models for the Whistleblowing and Retaliation Games 411
F.4.1 Automated Redirection of Whistleblowing Rewards 411
F.4.2 Anonymous vs Non-Anonymous Whistleblowing Models 420
F.5 Deferred Game-Theoretic Proofs 422
F.5.1 Proof of Theorem 10.16 422
F.5.2 Proof of Theorem 10.17 423
F.5.3 Proof of Theorem 10.18 424
F.5.4 Optimality of $\mathbb{M}_{1*}^{\mathbb{W}}$ 426
F.5.5 Deferred details for (t, n)-collusion 428

Bibliography **430**

LIST OF TABLES

6.1 Comparison of Themis to existing fair ordering protocols. An entry in **green** indicates that it is the best property among the protocols that we compare to. An entry in **red** indicates a significant shortcoming. **(1)** *Fair separability* (Definition 7.1) captures both notions (see Section 7.1). Our experiments in Section 7 show why this notion is significantly weaker than the one achieved by Themis and is likely not enough to handle adversarial order-manipulation in real-world settings; **(2)** In Wendy, if honest local clocks are far apart (e.g., several seconds apart), then their fairness definition will never apply and therefore no fairness guarantees can be provided; **(3)** We show that κ-differential-order-fairness is simply a reparameterization of batch-order-fairness (see Section 9.8 and Appendix D.3); **(4)** Since we consider a total ordering in Themis, transactions within a batch are ordered contiguously for execution (see Section 6.3.3 and Section 6.4); **(5)** The fairness properties of both Aequitas and Themis are parameterized by γ but even for the weakest variant, i.e., corresponding to $\gamma = 1$, they are still stronger than those of Wendy [170] and Pompē [259]. For $\gamma = 1$, Aequitas and Themis both require $n \geq 4f+1$ but also work with $n \geq 3f + 1$ if only crash-faults are present. 123

7.1 Number of frontrunnable pairs for fair separability and order-fairness for several settings. The first line denotes to our example setting with 5 geo-distributed AWS nodes. For the other rows, we randomly sample n-sized committees from the Wonderproxy dataset [251]. We chose $n = 20, f = 4$ and $n = 100, f = 24$ for the second and third row respectively. 167

D.1 Average ping times (in ms) between different AWS servers 384

LIST OF FIGURES

4.1 Static Finalization Algorithm 81
4.2 Simple Finalization Example 82

5.1 Permissionless Order-Fair Protocol Design 97
5.2 Pictorial representation of $\Pi^{\kappa,T}_{\text{mod}}$ 101
5.3 Protocol $\Pi^{\kappa,T}_{\text{mod}}$ 102
5.4 Pictorial representation of $\Pi^{\kappa,d}_{\text{multi}}$ 105
5.5 Protocol $\Pi^{\kappa,d}_{\text{multi}}$ 106

6.1 Leader Proposal Algorithm 140
6.2 Update Algorithm 142
6.3 Complete Description of Themis 143
6.4 FairFinalize Algorithm 146
6.5 Performance comparison of Themis and Hotstuff in both the same-datacenter and geo-distributed settings. 153

7.1 Comparison of the fraction of transaction pairs correctly ordered by different fairness notions as a function of r. 161
7.2 Comparison of the number of transactions in a cycle of specific lengths. A value of 1 means that there is no cycle. 162
7.3 Adversarial influence on transaction ordering 169

8.1 Example of equilibrium signaling functions for $g = 10$ and $\Delta = 0.1$. Timestamps are normalized so that $t_2 = 0$. The blue function is the equilibrium signal $\pi_1(v) - t_1$ for bidder 1 as a function of the valuation. The red function is the equilibrium signal $\pi_2(v) - t_2$ for bidder 2 as a function of the valuation. 199

9.1 *Proof of complete knowledge* (PoCK) setting. A PoCK is a proof of knowledge, but has two additional requirements: (1) $\mathcal{P}$ can only execute the proof successfully by accessing a special local *resource* $\mathcal{R}$; and (2) $\mathcal{P}$ must have access to the witness sk as it is transmitted to $\mathcal{R}$ over a plaintext *eavesdropping channel* $\mathcal{E}$. Green lines / boxes in the figure indicate entities local to / within the trust domain of $\mathcal{P}$, while blue lines / boxes indicate those outside. 221
9.2 SGX-PoCK Protocol Description. 249
9.3 ASIC-ZKPoCK protocol. The protocol executes over n rounds. $\mathcal{V}$ runs PoKAccept in each round to check that $\mathcal{P}$ given a valid Σ transcript (a_i, c_i, s_i) and that $\mathcal{P}$ has run within time bound τ. After n rounds, $\mathcal{V}$ runs puzAccept^y to check more than y out of the n PoW puzzle solutions $\pi_1, ..., \pi_n$ are correct with respect to the difficulty specified by d. 258
9.4 Protocol notation (left) and parameters (right) 259

9.5 ASIC-ZKPoCK allows for practical parameters to achieve overwhelming probability of completeness by an ASIC (hashrate 154 TH/s) and negligible probability of success by an adversary with 10,000 state-of-the-art CPUs (hashrate 72 MH/s). n denotes number of rounds. Here, prover is required to compute more than $y = \lfloor n/2 \rfloor$ valid PoW puzzle solutions. The black line represents a parameterization of $d = 10^{14}$, for which adversarial success probability is negligible but that of ASIC is close to 1. 264
9.6 Lines of code (LOC) for SMACK and gas costs for various operations. As of Oct 2023, the cost for 100,000 gas is around $1.55. . 264
9.7 ASIC-ZKPoCK system architecture. The encircled numbers correspond to steps in a protocol execution. Our CK Prover consists of a (modified) pool mining software and an ASIC miner. Our CK Verifier is implemented as a Smart Contract on Ethereum mainnet. 267

10.1 The unframability game for a whistleblowing protocol $\mathcal{W}_{\mathcal{F}}$ and a functionality $\mathcal{F}$. 297

E.1 $\mathcal{G}_{\mathsf{SGX}}$ global functionality from [209]. Starred operations are reentrant activation points. 388

F.1 The soundness game $\mathbf{G}^{\text{sound}}_{\mathcal{F}}$ for a functionality $\mathcal{F}$. 400

CHAPTER 1

INTRODUCTION

Distributed systems have been studied by computer science researchers and practitioners for the past five decades. In a nutshell, a *distributed* system uses several machines to offer the functionality of a single centralized logical entity but typically with better scalability or higher resilience to failures and malicious actors. Fault-tolerant distributed systems are commonplace in modern infrastructure—most web applications, banking, smart energy grids, reservation systems, and more.

A core tenant of "classical" distributed systems is that its executors, i.e., entities in charge of running the system need to be known to each other. Users may of course be numerous and unknown in advance. This perception, however, has changed in the past decade or so. Broadly, this dissertation pertains to this changing landscape, the applications it makes possible, and most importantly, its downstream effects on the security and robustness of our systems.

1.1 Origins

In late 2008, the pseudonymous Satoshi Nakamoto circulated an innocuous-looking paper [197] introducing Bitcoin, a novel digital currency that did not require a centralized authority or government for coordination. The protocol allowed *anyone* in the world to take part not just as a user but also as an *executor* in charge of maintaining the system and preventing attacks. In essence, Bitcoin was secured against attackers through *public decentralized coordination*, marking its vision of a currency "of the people, by the people, and for the people." In the

years that followed, Bitcoin would usher in an era of "cryptocurrencies", with 1000s of currencies totaling a $3 trillion USD market cap.

Still, something grander, at least from an academic perspective, lurked underneath—a radically new perspective on distributed systems. Bitcoin introduced the (now buzzwordy in its own right) idea of *blockchain*. A few years later in 2015, the launch of Ethereum, the second cryptocurrency, showed the generality of this idea—blockchains could power general-purpose applications beyond digital currencies with the same decentralized coordination that secured Bitcoin.

1.2 Blockchain Research

In the decade since Ethereum, blockchains have become a new class of systems, often dubbed *decentralized* systems to underscore their central characteristic. Arguably, blockchains have ushered in a new era in distributed systems and cryptography research. Major advancements in consensus protocols, zero-knowledge proofs, digital signatures, multi-party computation, timed-cryptography, and automated market makers can be attributed in part to their applicability to blockchains.

Much of this research effort has focused on building new tools or improving the efficiency and scalability of existing ones. However, as blockchain applications have grown beyond cryptocurrencies, it has become increasingly clear that we not only need to build secure protocols, but also think more carefully about *what* blockchain security means. This dissertation presents one such exploration.

1.3 This Dissertation

This dissertation begins with a simple, yet intentionally vague question without a clear answer. Arguably, this remains true even at the end of the dissertation.

What does security in the blockchain setting really mean?

As an overarching claim, security in the blockchain setting has new challenges. Solving them requires looking beyond traditional approaches to security. Blockchains operate in a radically new, highly adversarial environment and further come with built-in money (often billions of dollars of it) and self-executing permissionless governance. This typically means that any subtle protocol flaw or software bug could be immediately monetized by anonymous actors. As a consequence, blockchains fundamentally intertwine an assortment of fields—cryptography, distributed systems, game theory and mechanism design, macroeconomics, social choice and governance, and potentially many others. When designing secure blockchains and their applications, a cryptographer should no longer model only the relevant cryptography, a game-theorist should no longer focus purely on the game theory. The research in this dissertation is in part, influenced by the difficulty of designing secure protocols in this new adversarial blockchain environment.

At the same time, understanding blockchain security can also influence how we should go about designing secure systems more broadly. In fact, the highly adversarial blockchain setting makes an excellent sandbox both for designing more robust protocols and for discovering new adversarial threats.

This dissertation aims to illustrate, through examples, *why* an intersectional

viewpoint is required for blockchain security and what could happen otherwise. In the process, it questions established modeling assumptions in well-studied, longstanding security frameworks, and finds novel insights about classical security notions—many of which are of independent interest even beyond blockchains. In no way is this dissertation supposed to be a grand unifying theory of blockchain security—current research is far from that. Rather, the hope is that it serves as an example of why blockchain security may need to rethought from the ground up.

This dissertation shows how (1) new blockchain applications, and (2) new attacker capabilities can surface subtle new threats and provide valuable insights on classical security notions. It comprises of two parts as below:

1.3.1 Decentralized Finance and Transaction Ordering

Part I is broadly on blockchain transaction ordering, motivated by new decentralized financial applications housed on blockchains. In financial contexts, the order in which a user's trade (or transaction) is executed is extremely important since it can affect e.g., her profit. Yet, the consensus primitive, which underpins distributed systems including blockchains, does not prescribe how transactions need to be ordered. This under-specification leads to new, highly profitable order-manipulation attacks [92, 216] at the scale of $1 billion USD per year on real-world blockchain systems.

The work in this part introduces a new research direction on *order-fair* consensus protocols, which decentralize trust in the ordering, and provide strong temporal fairness guarantees on the way transactions are ordered. Along the

way, it also uncovers broader connections of order-fairness to questions in social choice theory. My initial work has spawned an active research direction on transaction ordering with 50+ direct followup works by the community that propose new protocols and definitions.

The content in this part was first published as the papers [161, 157, 158, 180].

1.3.2 Insights from New Attacker Capabilities

Part II shows two distinct examples of new attacker capabilities spelling problems for classical security frameworks.

1.3.2.1 Proofs of Complete Knowledge

The first example pertains to "knowledge" in cryptography, coercion-resistant voting, and blockchain governance. The security goal for many cryptographic protocols is based on a simple intuition: the honest user—who "knows" the secret key—should succeed, while the attacker—who doesn't know the key—should fail. But it turns out that this common model can be eroded [215, 144] by attackers nefariously using trusted execution environments (TEEs) or secure multi-party-computation (MPC)—the very tools employed for building secure protocols. These tools allow for selectively restricting access to the key in ways that can break protocol guarantees without detection.

With collaborators, my work [18, 19] further shows how TEEs can be used to enable a broad range of powerful new attacks: coercion in coercion-resistant voting, dark marketplaces to circumvent blockchain governance, undetectable

renting of account credentials and private keys, and breaking deniability in deniable messing.

Chapter 9 in this dissertation unifies the attacks by identifying a common root cause—a subtle insufficiency in the way cryptography characterizes *knowledge*. It shows that standard proof-of-knowledge primitive doesn't always demonstrate knowledge of the secret key, and develops new theory on *proofs of complete knowledge* (CK) to fix this issue. CK enables proving *unrestricted access* to the secret key, which is in fact *necessary* to prevent these attacks. The content in this chapter was first published as the paper [156].

1.3.2.2 Smart Collusion

The second example is on the usage of blockchain *smart contracts* as a coordination tool between attackers. Smart contracts are programs that run on blockchains—essentially, they can be used to build general-purpose decentralized applications.

Chapter 10 shows an example of using smart-contracts nefariously in order to circumvent accountability techniques in multi-party cryptography. Multi-party cryptography has extensive applications in blockchains and beyond. Most protocols make some form of honesty assumption on player quorums—i.e., the protocol works correctly as long as a majority of players act honestly. If enough players collude with each other dishonestly, protocol security may be broken. This assumption is especially concerning in a blockchain setting, where such coordinated dishonest behavior can often result in substantial financial gain.

Several approaches exist in the literature to mitigate the risk that many play-

ers act dishonestly. In general, they establish ways to hold players accountable for misbehavior. Chapter 10 shows how colluders can circumvent many of these approaches by colluding *smartly*—i.e., by using blockchain smart-contracts to facilitate their coordination. The content in this chapter first appeared as the paper [159].

Part I

Insights from New Applications:

Blockchain Transaction Ordering

A core functionality provided by blockchains is that of a "decentralized, general-purpose computer (or state machine)." Intuitively, a blockchain enables its (distributed set of) participants to agree on the current state (and its evolution) of this state machine. Here, updates to the state are carried out through *transactions* submitted to the blockchain.

Classical distributed systems research informs us on how to build distributed state machines that are tolerant to faults (both benign and malicious) [171, 64]. Subsequent chapters will introduce more details through the "state machine replication" (SMR) or "consensus" abstraction. Modern blockchains extend the abstraction by enabling replication of the state machine even when the set of participants is unknown and can change dynamically.

Arguably, the largest use case for blockchains today is *decentralized finance* (DeFi) systems. For financial systems, the order in which transactions are sequenced (and subsequently executed) is extremely important as it can affect e.g., a user's profit. Discussion on transaction ordering however is largely missing from classical distributed systems literature, owning in part to the fact that traditional applications typically did not care about ordering.

In turn, for building robust blockchain applications, developing a theory on transaction ordering is important. This dissertation part covers my work on blockchain transaction ordering, which involves revisiting the SMR abstraction.

Organization. Chapter 2 formally introduces *order-fair* consensus, which extends the standard consensus (i.e., state machine replication) primitive to provide stronger guarantees on *how* transactions are ordered. Chapter 3 constructs the Aequitas family of order-fair protocols. The content in these chapters

was first published with the title "Order-Fairness for Byzantine Consensus" at CRYPTO 2020 [161].

Chapter 4 introduces a new perspective on order-fairness through the lens of social choice and voting theory. Chapter 5 extends the order-fairness definitions and protocols to the *permissionless* setting—i.e., where protocol participants are dynamic and identities are unknown. The content in these chapters was first published with the title "Order-Fair Consensus in the Permissionless Setting" at APKC 2022 [157].

Chapter 6 introduces Themis, an order-fair protocol built with practical performance requirements in mind. Along the way, it also unlocks deeper connections with the theory of order-fairness. The content in this chapter was first published with the title "Themis: Fast, Strong Order-Fairness in Byzantine Consensus" at ACM CCS 2023 [158].

Chapter 8 looks beyond order-fair consensus with a focus on mitigation of negative externalities of first-in-first-out ordering. The content in this chapter was firts published with the title "Buying Time: Latency Racing vs. Bidding for Transaction Ordering" at AFT 2023 [180].

CHAPTER 2

ORDER-FAIR CONSENSUS

Decades of research in both cryptography and distributed systems has extensively studied the abstraction of state machine replication (SMR), also known as Byzantine consensus[1]. The consensus abstraction is also the underlying bedrock of blockchain design. At a high level, the goal of a consensus protocol is for a set of nodes to agree on an ever-growing, linearly ordered log of messages (transactions in the blockchain context). Two properties need to be satisfied by such a protocol: (1) *Consistency* - all honest nodes must have the same view of the agreed upon log — that is, they must output messages in the same order; and (2) *Liveness* - messages submitted by clients are added to the log within a reasonable amount of time.

Unfortunately, neither consistency nor liveness says anything about the *actual ordering* of transactions in the final log. A protocol that ensures that all nodes agree on the same ordering is deemed consistent regardless of how the ordering is generated. This leaves room for the definition to be satisfied even if a single adversarial node directly chooses the actual transaction ordering.

As we shall see, this under-specification in the consensus abstraction creates significant problems especially in the context of blockchain financial applications. This chapter introduces transaction-*order-fairness* as a new property for consensus protocol design which seeks to *decentralize* the process of finalizing the transaction ordering and curtail adversarial influence.

The content in this chapter was initially published at CRYPTO 2020 as the

[1]The terms *state machine replication* and *consensus* will be used interchangeably.

work "Order-Fairness for Byzantine Consensus" [161].

2.1 Order-manipulation Attacks and the Importance of Fair Transaction Ordering

The need for a notion of fair transaction ordering is immediately clear when looking at financial systems. On blockchains today, the deployment of financial applications has attracted substantial interest in recent years. As of May 2025, over 100 billion USD is locked within this *decentralized finance* or DeFi ecosystem. Here, the execution order can determine the validity and/or profitability of a given transaction. In many scenarios, *when* a transaction is executed is even more important than *whether* it was executed at all; execution in an unfavorable order can result in unnecessary losses for the user. Simple illustrative examples are given below:

- (Example #1: Ordering affects validity). Suppose that Bob has \$0, and two transactions are initiated: tx_0, which sends \$5 from Alice to Bob, and tx_1, which sends \$5 from Bob to Carol. If tx_0 is sequenced before tx_1, then both transactions are valid; the opposite ordering invalidates tx_1. This means that an adversary who controls the transaction ordering can potentially invalidate user transactions.
- (Example #2: Ordering affects profitability). Suppose that Alice wishes to buy 1 share of company ABC at the current price of \$100. In typical financial markets, the transaction order determines the price at which a user's trade executes; *buy* orders create an upward pressure on the price while *sell* orders create a downward pressure. An adversary Mallory who con-

trols the transaction ordering can now employ a simple *sandwich* strategy as follows: Mallory places a buy-order for 1 share of ABC right before and a corresponding sell-order right after Alice's transaction. What happens now is that Mallory's buy-order will execute at \$100, increasing the execution price for Alice's transaction (say to \$101). The \$1 difference will be captured as *pure profit* by Mallory through the subsequent sell-order.

Manipulation of transaction ordering is a well known phenomenon on Wall Street [173] but recent work has shown it to also be commonplace in consensus-based systems such as permissionless blockchains. Prior to the publication of the content in this chapter at CRYPTO 2020, seminal work by Daian et al. [92] reported extensively on rampant adversarial manipulation of transactions in the Ethereum network [111] by bots extracting upwards of USD 6M in revenue from unsophisticated users. Since then, while DeFi has continued to gain popularity, a long line of work [109, 166, 260, 216, 248, 249, 200] has further demonstrated the rise of adversaries extracting profit at the expense of regular users by manipulating the ordering and inclusion of transactions in DeFi applications. We will refer to these as *order-manipulation attacks*. Order-manipulation has been shown to be extraordinarily lucrative—often estimated at upwards of \$1B annually—and in fact has spurred competition between players who seek to capture the profits for themselves [87, 118, 123]. It also has the potential to create systemic bribery threats [185], and even fundamental protocol instability [92, 166].

Feasibility of order-manipulation in existing consensus designs. Order manipulation is possible in existing protocols largely because the formal properties required from SMR or consensus protocols place *no restriction on how transactions are ordered*. Neither consistency nor liveness enforces any relationship between

the order in which transactions arrive in the network and their final ordering in the log. Most popular consensus protocols used in practice (e.g., [64, 183, 256]) follow a *leader*-based design where a designated leader node is designated with the task of ordering and batching user transactions into a block. However, this means that an adversarial leader may choose to propose transactions in any arbitrary order, including one that is most profitable for itself. Moreover, this is true even for protocols in the "permissionless" setting (such as Bitcoin and Ethereum); here, the current miner or validator fully controls the inclusion and ordering of transactions within a block that it creates. In a sense, while a consensus protocol can be thought as decentralizing trust in the execution of a state machine, *transaction ordering remains centralized* albeit ephemerally.

Comparison to validity in Byzantine agreement. Beyond its critical practical importance, we believe that order-fairness is a key missing theoretical concept in existing consensus literature. To underscore this point, consider Byzantine agreement [171], or *single-shot* agreement, another well-studied problem in consensus literature. For Byzantine agreement, each node starts with a single value within a set $\mathcal{V}$. The goal is for all nodes to agree on the same value. *Validity* now requires that if all honest nodes start with the same value v, then the agreed upon value should also be v.

The property of *order-fairness* is a natural analog of validity formulated for the consensus problem, i.e., extension of Byzantine agreement to multiple rounds. If all honest nodes start with the belief that a transaction tx_1 precedes another transaction tx_2, by natural analogy with validity, the final output log should sequence tx_1 before tx_2. Consequently, we maintain that *order-fairness* is a natural property of independent theoretical interest in the consensus litera-

ture.

2.2 Preliminaries and Model

In this section, we describe the general execution framework that we will use for expressing and analyzing consensus protocols. To define the state machine replication problem in an unconstrained setting, we adopt an approach like that of Pass and Shi [207, 208] and Chan et al. [68]. We focus on the "permissioned" setting — where the number of consensus nodes n, as well as their identities, is known a priori to all participants. While arbitrary clients can send messages to these nodes, only a fixed set of nodes will take part in the consensus protocol. We are also interested in protocols for several network settings (e.g. synchronous, partially synchronous, and asynchronous) and define constrained environments for these settings by imposing restrictions that an adversary must respect.

2.2.1 Protocol Execution Model

Interactive Turing Machines (ITMs). To model protocol execution, we adopt the widely used Interactive Turing Machine (ITM) approach rooted in the Universal Composability framework [57]. Informally, a protocol details how nodes interact with each other where each node is represented by an Interactive Turing Machine. As standard practice in cryptography literature [57, 61, 59], we use an environment $\mathcal{Z}(1^\kappa)$ (where κ is the security parameter) to direct the protocol execution. The environment $\mathcal{Z}$ can be thought of to represent everything that is

not defined by the protocol in consideration. $\mathcal{Z}$ is also responsible for activating nodes as either *honest* or *corrupt*, providing messages as inputs to nodes, and delivering messages between nodes. This is useful to model systems where protocol inputs may come from external applications and protocol outputs may be used by external applications. To communicate with others, a node sends a message to the environment, which is then relayed to other nodes as appropriate by the environment. Honest nodes follow the protocol description while corrupt nodes are assumed to be controlled by an adversary. This adversary, denoted by $\mathcal{A}$, is able to read all inputs/messages sent to corrupt nodes and can set all outputs/messages to be sent. The adversary also decides when messages sent over the network get delivered, of course subject to any network assumptions.

Rounds. We assume that the environment $\mathcal{Z}$ maintains a global clock. The clock is a global functionality [59] that contains a simple monotonic counter which can be updated adversarially by the environment. Informally, "global" means that the clock functionality exists in the system regardless of the analyzed protocol. This modeling choice follows from Canetti et al. [60]. Whether this clock is visible to protocol nodes depends on specific network settings. In synchronous settings, this clock is visible to all nodes[2]. In the synchronous setting [103], we can therefore model protocol execution in discrete time steps or rounds. At the start of each round, each node receives a set txs of transactions from the environment $\mathcal{Z}$. Transactions are assumed to be submitted by clients, but using the environment abstraction avoids having to model clients explicitly. Rather, the environment is in charge of providing transactions as input to the

[2]In [60], it is emphasized that honest parties do not talk directly to the clock functionality. We can circumvent this restriction by having the environment send the current time counter to each node as input every round.

nodes. Furthermore, at the end of each round, each node outputs an ordered log LOG to $\mathcal{Z}$ which intuitively represents the list of transactions ordered by the node so far. We assume that $\mathcal{Z}$ signals the start of a new round to each node.

Rounds in the partially synchronous setting [105] work similarly to the synchronous setting.

In the asynchronous setting [45], we assume that a global clock still exists in the environment. Except now, the clock is not accessible to the protocol nodes. The environment $\mathcal{Z}$ can provide user transactions and communication messages to nodes at any time. Without loss of generality, since protocol nodes cannot read the global clock, we can assume that the clock counter is incremented every time $\mathcal{Z}$ provides new transactions or delivers messages. Note that, we use the notion of rounds in an asynchronous setting merely as a tool for our analysis. It serves no purpose in the actual protocol and any protocol that works in the asynchronous setting should not rely on the current time. Throughout the paper, we may use the terms "time" and "round" interchangeably.

Notational conventions. We use κ to denote the security parameter. $\mathcal{N}$ denotes the set of protocol nodes. For a protocol Π, $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ represents the random variable for all possible execution traces of Π w.r.t. adversary $\mathcal{A}$ and environment $\mathcal{Z}$. The possible executions arise from any randomness used by honest nodes, adversarially controlled nodes, and the environment. Any view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ is a fully specified instance of an execution trace. That is, a particular view can be thought of as the joint view of all nodes (including all inputs, outputs, random coins etc.) during an execution. We use $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ to denote randomly sampling an execution. $|\mathsf{view}|$ de-

notes the number of rounds in view.

A function $\mathsf{negl}(\cdot)$ is *negligible* if for every polynomial $p(\cdot)$, there exists a constant $\kappa_0 \in \mathbb{N}$, such that $\mathsf{negl}(\kappa) \leq \frac{1}{p(\kappa)}$ for all $\kappa \geq \kappa_0$. We use $\mathsf{negl}(\kappa)$ to denote a function that is negligible as a function of κ.

Corruption Model. Since we are concerned only with the permissioned setting, we consider environments $\mathcal{Z}$ that do not spawn any more nodes after an initial spawn. In particular, $\mathcal{Z}$ spawns a set of nodes, numbered from 1 to n without loss of generality at the start. It never spawns any additional nodes. At any point, $\mathcal{A}$ can ask $\mathcal{Z}$ to corrupt a particular node for which $\mathcal{Z}$ sends a `corrupt` signal to that node. When this happens, the internal state of that node gets exposed to $\mathcal{A}$ and $\mathcal{A}$ henceforth fully controls the node. $\mathcal{A}$ gets full control over all corrupt nodes, including the ability to control their messages and outputs.

A node is said to be *honest* in a given view if it is never under adversarial control, otherwise it is said to be *corrupt* or *byzantine*. Note that once a node is corrupted, it cannot become honest at a later point. In our general model, we assume that the adversary can corrupt nodes dynamically. That is, nodes can be corrupted at any point during the protocol's execution. We use a corruption parameter f to denote the maximum number of nodes that $\mathcal{A}$ can corrupt.

Communication and Network Model. As mentioned before, the environment $\mathcal{Z}$ provides transactions sent by users as inputs to nodes and also handles communication between nodes. We assume that a node can broadcast a message to any subset of recipients through an authenticated channel. The environment $\mathcal{Z}$

delivers any broadcast messages to its recipients at the start of a round, along with any new transaction inputs from users for that round. Furthermore, we assume that the adversary $\mathcal{A}$ cannot modify messages sent by honest nodes but can reorder or delay messages, possibly constrained by the specific setting. We also assume the existence of a public-key infrastructure (PKI) — each node has a public key registered with the PKI. The public key of a node is given to all nodes on initialization by the environment. Note that in a PKI, digital signatures can be used to prove the identity of the message sender. Digital signatures can be realized from a PKI, as a global signing functionality $\mathcal{G}_{\text{sign}}^{\Sigma}$ (parameterized by a signature scheme Σ). We refer the reader to [208] for further details. For our paper, we can abstract away the actual implementation of signatures since in a PKI, without adding any communication overhead, we can assume that $\mathcal{Z}$ simply reveals the identity of the sender when forwarding a message to the recipient(s). We note that this is equivalent to working in the $\mathcal{G}_{\text{sign}}^{\Sigma}$-hybrid world where nodes query the global functionality $\mathcal{G}_{\text{sign}}^{\Sigma}$ when they need to sign messages.

We differentiate between two networks in our model - an *internal* network for communication between nodes and an *external* network for how external users send transactions to nodes. We emphasize that $\mathcal{A}$ is only in charge of scheduling message delivery for the internal network. The external network may reside in other parts of the application (not relevant to the consensus protocol) and is managed by $\mathcal{Z}$ (and possibly by some other network adversary). However, we may abstract specific timing properties from the external network to prove our results.

Depending on the network delay properties, we consider the synchronous

setting [103] (where the network delay bound is known), the partially synchronous setting [105] (where the network delay bound is finite but unknown), and the asynchronous setting [45] (where the network delay is unbounded).

2.2.2 Execution Environments

Network Assumptions. First, we formally define the different network assumptions for both the external and internal networks. We assume that clients submit transactions to the system by sending them to all the nodes. As mentioned before, we do not explicitly model clients, but rather have transactions input by the environment. Any network assumptions are modeled as restrictions imposed on the environment.

External Network. The external network models the communication channel between the system users and the protocol nodes. Any assumptions on the external network can be thought of as assumptions on how the environment acts. By a synchronous external network, we mean that any transaction that is received (from the environment) by a node reaches all other nodes within a known time. This is formally defined in Definition 2.1.

Definition 2.1 (External Synchronous Setting)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects $\Delta_{\mathsf{ext}} = (\mathsf{full}, \delta)$ ext-synchrony w.r.t. a protocol Π if for every $\kappa \in \mathbb{N}$ and view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following conditions hold: (1) $\mathcal{Z}$ provides δ as a public parameter to all nodes upon spawning; (2) If $\mathcal{Z}$ provides an input message m to a node as input in the txs set at time t, then at any time $t' \geq t + \delta$, all other nodes will also have received message m as input.

For the partially synchronous setting, we assume that the delay bound δ exists but is unknown to the nodes. Partial synchrony in the external network is defined similar to the synchronous setting, except now, $\mathcal{Z}$ does not provide the parameter δ to the nodes upon spawning. We use $\Delta_{\mathsf{ext}} = (\mathsf{partial}, \delta)$ to denote the partially synchronous setting. For the asynchronous setting, we only assume that transactions are not dropped by the network — they eventually get delivered to all the nodes. However, we make no assumptions on the actual delivery time. We use $\Delta_{\mathsf{ext}} = (\mathsf{none}, \infty)$ to denote an asynchronous external network.

Internal Network. The internal network represents the network between nodes and is usually the standard network considered for consensus problems. For the internal network, synchrony is the assumption that any message sent by a node reaches the recipient(s) in a known, finite time δ. Definition 2.2 formalizes this synchrony assumption. Recall that $\mathcal{Z}$ delivers messages only at the start of a round.

Definition 2.2 (Internal Synchronous Setting)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects $\Delta_{\mathsf{int}} = (\mathsf{full}, \delta)$ int-synchrony w.r.t. a protocol Π if for every $\kappa \in \mathbb{N}$ and view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following conditions hold: (1) $\mathcal{Z}$ provides δ as a public parameter to all nodes upon spawning; (2) If an honest node sends a message at time t, then at any time $t' \geq t + \delta$, all recipient(s) will have received the message.

The partially synchronous and asynchronous settings for the internal network are defined similarly to the corresponding notions for the external network. We use $\Delta_{\mathsf{int}} = (\mathsf{partial}, \delta)$ and $\Delta_{\mathsf{int}} = (\mathsf{none}, \infty)$ to denote a partially synchronous

internal network and an asynchronous internal network respectively.

Other network nomenclature. We say that the network is *completely synchronous* (resp. *completely asynchronous*) if both the external and the internal network are synchronous (resp. asynchronous). We say that the external network is *instant synchronous* if $\Delta_{\mathsf{ext}} = (\mathsf{full}, 0)$. We use $\mathsf{not\text{-}async}$ to denote both the synchronous setting (full) and the partially synchronous setting $(\mathsf{partial})$.

We formalize the permissioned setting next.

Permissioned Setting. We can express the "permissioned" or "classical" environment by placing the following constraints on $(\mathcal{A}, \mathcal{Z})$: In the permissioned setting, we require that the environment $\mathcal{Z}$ spawn all nodes upfront and not spawn any new nodes during the protocol execution. Furthermore, all nodes know the identity of all other nodes in the protocol. Without loss of generality, we can assume that the initial nodes spawned by $\mathcal{Z}$ are numbered from 1 to n. We define such a permissioned environment in Definition 2.3.

Definition 2.3 (Classical Permissioned Environment)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$-classical execution w.r.t. a protocol Π if it respects Δ_{int} int-synchrony, Δ_{ext} ext-synchrony and for every $\kappa \in \mathbb{N}$ and view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following conditions hold: (1) $\mathcal{Z}$ spawns a set of nodes numbered from 1 to n at the start of the protocol and never spawns any nodes later; (2) $\mathcal{Z}$ does not corrupt more than f nodes; (3) $\mathcal{Z}$ provides all nodes the parameters (n, f) upon spawning; (4) $\mathcal{Z}$ also provides all nodes any other public parameters upon spawning. This includes the node identities as well as any public keys.

Notation. For all constraints on $(\mathcal{A}, \mathcal{Z})$, when the context is clear, we may choose to exclude which protocol we are referring to. For example, we may simply write $(\mathcal{A}, \mathcal{Z})$ respects $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$-classical execution. For the remainder of the paper, we will only consider $(\mathcal{A}, \mathcal{Z})$ that respect classical execution.

2.2.3 The State Machine Replication Abstraction

In the state machine replication or consensus problem, a set of nodes try to agree on a growing, linearly ordered log. At the start of each round, $\mathcal{Z}$ may provide a set txs of transactions to protocol nodes. We assume that the transactions input by $\mathcal{Z}$ are unique. At any time, nodes may also choose to deliver transactions by outputting a log of transactions LOG to $\mathcal{Z}$. The LOG can be thought of as a totally ordered sequence where each element is an ordered set of transactions. We refer to the set of transactions at an index of the LOG as a "block". The LOG represents the set of transactions committed by a node so far.

Transaction nomenclatures. When discussing the trajectory of a transaction, several related terms are used in literature. We say that a transaction tx is *received* by a node when it is given as input to the node by $\mathcal{Z}$. A transaction tx is *delivered* or *committed* or *output* by a node when it is included in a LOG output by the node to $\mathcal{Z}$.

Notation for the ordered log. Suppose that $\mathcal{T}$ denotes the space of all possible transactions. Let LOG_i represent the most recent log output by node i to the environment i.e. LOG_i represents the totally ordered list of transactions that node i has delivered so far.

For two logs LOG and LOG', we define a relation $\preceq$ which intuitively signifies a "prefix" notion. $\mathsf{LOG} \preceq \mathsf{LOG}'$ stands for "LOG is a prefix of LOG'". We assume that for any x, we have $x \preceq x$ and $\emptyset \preceq x$. $\mathsf{LOG}[p]$ denotes the p^{th} element in LOG. $\mathsf{LOG}(m)$ denotes the number p such that $\mathsf{LOG}[p]$ contains m.

The security of a state machine replication protocol can now be defined as follows:

Definition 2.4 (Security of state machine replication [208])**.** We say that a protocol Π satisfies consistency (resp. $(T_{\text{warmup}}, T_{\text{confirm}})$-liveness) w.r.t. $(\mathcal{A}, \mathcal{Z})$ if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, the consistency (resp. $(T_{\text{warmup}}, T_{\text{confirm}})$-liveness) property is satisfied except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow\!\!\$\ \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ where $\mathsf{negl}(\cdot)$ is negligible in κ. For a particular view, we define the properties below:

- **(Consistency)** A view satisfies consistency if the following holds:
  - *Common Prefix.* If an honest node i outputs LOG to $\mathcal{Z}$ at time t and an honest node j outputs LOG' to $\mathcal{Z}$ at time t', then it holds that either $\mathsf{LOG} \preceq \mathsf{LOG}'$ or $\mathsf{LOG}' \preceq \mathsf{LOG}$.
  - *Future Self Consistency.* If a node that is honest between times t and t', outputs LOG at time t and LOG' at time $t' \geq t$ to the environment $\mathcal{Z}$, then it holds that $\mathsf{LOG} \preceq \mathsf{LOG}'$.
- **(Liveness)** A view satisfies $(T_{\text{warmup}}, T_{\text{confirm}})$-liveness if the following holds: At a time t such that $T_{\text{warmup}} < t \leq |\mathsf{view}|$, if an honest node either received a transaction m from $\mathcal{Z}$ or output m in its log to $\mathcal{Z}$, then for any honest node i and any time $t' \geq t + T_{\text{confirm}}; t' \leq |\mathsf{view}|$, it holds that m is in the log output by node i at time t'.

Here, T_{confirm} and T_{warmup} are polynomial functions in the security parameter κ, the number of nodes n, the corruption parameter f, the maximum network delay bounds as defined in Δ_{ext} and Δ_{int} (for synchronous and partially synchronous networks only), as well as the actual network delay. T_{warmup} is the protocol's warmup time, until which point liveness need not be satisfied. T_{confirm} is the maximum time it takes for a transaction (input after the warmup time) to be delivered by all honest nodes.

Note that the actual network delay is required as a parameter only for completely asynchronous networks. When the network is not asynchronous, the actual network delay is bounded by the maximum delay parameter. In such cases, the polynomials T_{confirm} and T_{warmup} can be bounded by replacing the actual network delay by the appropriate delay bound. While this is true, synchronous protocols where T_{confirm} does not depend on the maximum delay bound but rather on the actual network delay can confirm transactions much faster. The term *responsive* [205] is used to refer to such protocols.

Liveness in asynchronous networks. In the asynchronous setting, we assume that the network delay is an *unbounded* polynomial [205] in the security parameter. Equivalently, there does not exist a concrete polynomial T_{confirm} that serves as the liveness bound. Rather, we require that as long as the environment eventually delivers messages, honest nodes eventually include transactions in their output logs. Note that since the environment eventually delivers all messages before the protocol execution finishes, all transactions input by the environment should be eventually delivered by a *live* protocol. We define *asynchronous* or *eventual* liveness below.

- **(Asynchronous / Eventual Liveness)** A view satisfies $(T_{\text{warmup}}, \text{none})$-eventual liveness, or simply T_{warmup}-eventual liveness, if the following holds: At a time t such that $t > T_{\text{warmup}}$, if an honest node either received a transaction m from $\mathcal{Z}$ or output m in its log to $\mathcal{Z}$, then for any honest node i, at the end of protocol execution, it holds that m is in the log output by node i.

Weak liveness. The standard definition of liveness of a transaction tx (from Definition 2.4) is independent of what happens in the rest of the protocol's execution. Sometimes however, it may be enough for a protocol to be live only if transactions continue to be received by the system. For example, a transaction tx will only be delivered if there is some transaction that is received by all nodes sufficiently after tx. Intuitively, later transactions will cause earlier ones to be "flushed out" of the system. We note that this subtle distinction between the two liveness definitions is rarely considered in the literature. We found that some leaderless protocols (i.e. those that are not based on a leader node) like the ones in [221, 23] implicitly ignore this distinction. Along similar lines, we define a weaker version of conventional liveness, which we call "weak-liveness." Despite the technical difference, we think that it should be acceptable in most real world systems. For a particular view, we define weak-liveness below.

- **(Weak Liveness)** A view satisfies $(T_{\text{warmup}}, T_{\text{confirm}})$-weak-liveness if the following holds: Suppose that at a time t such that $t > T_{\text{warmup}}$, an honest node either received a transaction m from $\mathcal{Z}$ or output m in its log to $\mathcal{Z}$. Let T be a set built recursively as follows: (1) Add m to T; (2) For $m_0 \in \mathsf{T}$, add to T, all transactions m_0' that were received by at least one honest node before m_0. Now if another transaction m' was received at time t' and is such

that it was first received by a node after all nodes received all transactions in T, then for any honest node i and any time $t'' \geq t' + T_{\text{confirm}}; t'' \leq |\mathsf{view}|$, it holds that m is in the log output by node i at time t''.

We also define *weak eventual liveness*, which provides a version of weak liveness for the asynchronous setting.

- **(Weak Eventual Liveness)** A view satisfies $(T_{\text{warmup}}, \mathsf{none})$-weak-eventual-liveness or simply T_{warmup}-weak-eventual-liveness if the following holds: Suppose that at a time t such that $t > T_{\text{warmup}}$, an honest node either received a transaction m from $\mathcal{Z}$ or output m in its log to $\mathcal{Z}$. Let T be a set built recursively as follows: (1) Add m to T; (2) For $m_0 \in \mathsf{T}$, add to T, all transactions m_0' that were received by at least one honest node before m_0. If another transaction tx′ was first received by nodes after all nodes received all transactions in T, then for any honest node i, at the end of protocol execution, it holds that m is in the log output by node i.

For all of the liveness properties, we say that a protocol Π satisfies the property if there exists a negligible function $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, the property is satisfied except with $\mathsf{negl}(\kappa)$ probability over the choice of $\mathsf{view} \leftarrow\!\!\$\ \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ where $\mathsf{negl}(\cdot)$ is negligible in κ.

2.3 Defining Fair Ordering

We formally define fair ordering in this section. As it turns out, providing a definition that is achievable by protocols, yet intuitive, is not trivial. Some natu-

ral definitions are not achievable except under strong assumptions. We use this section to also go through these definitions that led to our final definition.

2.3.1 Send-Order-Fairness

A strawman approach is to require ordering to be in terms of when transactions were *sent* by clients. For instance, if a transaction tx_1 was sent by a client before another transaction tx_2 (possibly by another client), then tx_1 should appear before tx_2 in the agreed upon log. Not surprisingly, this can lead to several problems, which we briefly describe below.

Most importantly, there needs to be a trusted way to timestamp a transaction at the client side. In other words, there needs to be a trusted way for a client to prove that her transaction was generated at time t. Intuitively, this would require the presence of some trusted party to attest to the accuracy of the generated timestamp. Trusted execution environments (TEEs), e.g., Intel SGX [151], are a potential way to provide such a trusted timestamp as they provide protection for client-side software from a untrusted host (i.e., the client). Unfortunately, current TEE implementations cannot provide any notion of trusted global time. We note that retrieving time from a trusted source is not enough since an untrusted host could arbitrarily delay incoming timestamps.

Moreover, apart from assuming such a timestamping service, network synchrony is also required to ensure that a transaction is not arbitrarily delayed (either by a offline user or by a malicious network adversary). To see why, observe that a user could always generate a trusted timestamp and then simply hold on to the attested transaction until a favorable time. If the external network (be-

tween the users and protocol nodes) is asynchronous or partially synchronous, then there is no way to distinguish whether a transaction was delayed by the network or simply withheld by the user. An asynchronous network would in fact require protocol nodes to wait an unbounded amount of time to ensure that no transaction should be ordered earlier. Consequently, for send-order-fairness to be feasible, it is imperative that the external network be synchronous.

Trusted client-side timestamps would enable a new design paradigm for time-sensitive systems (e.g., financial exchanges). We leave their design open for future work.

The challenges of send-order-fairness suggest it would be more prudent to define fair ordering in terms of when the consensus nodes actually *receive* transactions. Since every node might receive transactions at slightly different times, or in a slightly different order, care must be taken in formulating the definition. We introduce a natural notion next.

2.3.2 Receive-Order-Fairness

Intuitively, "receive order" means that the fair ordering is defined by looking at when *enough* nodes receive a particular transaction. For instance, if sufficiently many nodes receive a transaction tx_1 before another transaction tx_2, then tx_1 must appear before tx_2 in the final log. This is formalized in Definition 2.5, where "sufficiently many" is parameterized using γ. We refer to γ as the order-fairness parameter.

Definition 2.5 (Receive-order-fairness)**.** For a view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, we define receive-order-fairness as follows:

- A view satisfies $(\gamma, T_{\text{warmup}})$ receive-order-fairness if the following holds: For any two transactions m and m', let η be the number of nodes that received both transactions between times T_{warmup} and $|\mathsf{view}|$. If at least $\gamma\eta$ of those nodes received m before m' from $\mathcal{Z}$, then for all honest nodes i, i does not deliver m' unless it has previously delivered m.

A protocol Π satisfies $(\gamma, T_{\text{warmup}})$ receive-order-fairness w.r.t $(\mathcal{A}, \mathcal{Z})$ if there is a negligible function $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, the order-fairness property is satisfied except with probability $\mathsf{negl}(\kappa)$ over a random choice of $\mathsf{view} \leftarrow\!\!\$\ \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$.

2.3.3 Condorcet paradox and the impossibility of fair ordering

The Condorcet paradox [88], or the "voting paradox", is a result in social choice theory that shows how some situations can lead to non-transitive collective voting preferences even if the preferences of individual voters are transitive. To illustrate how this applies to fair ordering, let us look at a simple example:

Example 2.6. Suppose that there are 3 nodes: A, B, and C. In the protocol execution, 3 transactions, $\mathsf{tx}_1, \mathsf{tx}_2$, and tx_3 are sent by clients to all the nodes.

- Node A receives transactions in the order $\mathsf{tx}_1, \mathsf{tx}_2, \mathsf{tx}_3$.
- Node B receives transactions in the order $\mathsf{tx}_2, \mathsf{tx}_3, \mathsf{tx}_1$.
- Node C receives transactions in the order $\mathsf{tx}_3, \mathsf{tx}_1, \mathsf{tx}_2$.

Now, 2 nodes (A and C) received tx_1 before tx_2, 2 nodes (A and B) received tx_2 before tx_3, and 2 nodes (B and C) received tx_3 before tx_1. It is easy to see that no

protocol can satisfy fair ordering for $\gamma \leq \frac{2}{3}$, since such a protocol would have to include tx_1 before tx_2; tx_2 before tx_3; and tx_3 before tx_1 in its final log.

Theorem 2.7 extrapolates this observation to a system with n consensus nodes.

Theorem 2.7. *Consider any $n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}}$ where Δ_{ext} is either (none, ∞) or $(\mathsf{not\text{-}async}, \delta_{\mathsf{ext}} \geq n)$. Let $\gamma \leq \frac{n-1}{n}$. If a consensus protocol Π satisfies $(T_{\mathrm{warmup}}, T_{\mathrm{confirm}})$-liveness w.r.t. all $(\mathcal{A}, \mathcal{Z})$ that respect $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$-classical execution, then it cannot also satisfy $(\gamma, T_{\mathrm{warmup}})$-receive-order-fairness (from Definition 2.5).*

Proof. Deferred to Appendix A.1. □

Notice that the result in Theorem 2.7 only requires properties from the external network, and is actually independent of the adversary. In other words, receive-order-fairness for an order-fairness parameter $\gamma \leq \frac{n-1}{n}$ is impossible to achieve *even when* there is no adversary.

Following the previous result, one would think that receive-order-fairness might still be possible for $\gamma = 1$. Unfortunately, a simple followup theorem shows this to be impossible in the presence of even a single corrupt node.

Theorem 2.8. *Consider any $n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}}$ where $f \geq 1$ and where Δ_{ext} is either (none, ∞) or $(\mathsf{not\text{-}async}, \delta_{\mathsf{ext}} \geq n)$. Let $\gamma \leq 1$. If a consensus protocol Π satisfies consistency and $(T_{\mathrm{warmup}}, T_{\mathrm{confirm}})$ liveness w.r.t. all $(\mathcal{A}, \mathcal{Z})$ that respect $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$-classical execution, then it cannot also satisfy $(\gamma, T_{\mathrm{warmup}})$ receive-order-fairness.*

Proof. Deferred to Appendix A.1. □

2.3.4 Environments that support receive-order-fairness

We find that the Condorcet paradox can be circumvented in a few ways by assuming specific network properties.

External synchrony assumption. The primary reason for the impossibility of fair-ordering is that different nodes may receive the same client transaction several rounds apart, resulting in non-transitive collective ordering. Synchrony in the external network can prevent any such non-transitive ordering. To elaborate, suppose that $\Delta_{\mathsf{ext}} = (\mathsf{full}, \delta)$ where $\delta \leq 1$ (e.g., an instant synchronous external network). Then, any client transaction that a node receives will reach all other nodes within 1 round. This implies that if some node receives transactions $\mathsf{tx}_1, \mathsf{tx}_2$ and tx_3 in that order, then no node can receive tx_3 before tx_1. It is now straightforward to see how this circumvents the Condorcet paradox.

Non-corrupting adversary and $\gamma = 1$. If the adversary does not corrupt any nodes, and its power is restricted to influencing network delays, we find that it is possible to achieve receive-order-fairness for $\gamma = 1$. In this setting, a single leader can receive the transaction orderings from individual nodes, and decide on a final ordering that preserves receive-order-fairness.

2.3.5 Towards weaker definitions for order-fairness

We give two natural relaxations of the original definition. The first is *approximate-receive-order-fairness* (or simply *approximate-order-fairness*) while the second is *batch-receive-order-fairness* (or simply *batch-order-fairness*). For

approximate-order-fairness, we only look at unfairness in the ordering of two transactions if they were received sufficiently apart in time. We emphasize that approximate-order-fairness only makes sense in synchronous and partially synchronous settings. On the other hand, for batch-order-fairness, we choose to ignore the ordering within a batch while considering fair ordering. Notably, this allows us to circumvent the Condorcet paradox by aggregating any transactions with non-transitive orderings into the same batch. This is reasonable to consider even in asynchronous environments.

First, we look at approximate-order-fairness. For a given view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, we define the property below.

Definition 2.9 (Approximate-Order-Fairness)**.** A view satisfies $(\gamma, T_{\text{warmup}}, \xi)$ approximate-order-fairness if the following holds: For any two transactions m and m', let η be the number of nodes that received both transactions between times T_{warmup} and $|\mathsf{view}|$. If at least $\gamma\eta$ of those nodes received m more than ξ rounds before m' from $\mathcal{Z}$, then for all honest nodes i, i does not deliver m', unless it has previously delivered m.

A protocol Π satisfies $(\gamma, T_{\text{warmup}}, \xi)$ approximate-order-fairness w.r.t $(\mathcal{A}, \mathcal{Z})$ if there is a negligible function $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, the above property is satisfied except with probability $\mathsf{negl}(\kappa)$ over a random choice of $\mathsf{view} \leftarrow\!\!\$\ \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$.

Quickly, we notice that a protocol which satisfies $(T_{\text{warmup}}, T_{\text{confirm}})$-liveness, also satisfies $(1, T_{\text{warmup}}, \xi)$ approximate-order-fairness for any $\xi \geq T_{\text{confirm}}$. Clearly, if a transaction tx_2 was received after tx_1 was delivered by all nodes, then tx_2 will be delivered after tx_1. Moreover, we also find that if $\xi < T_{\text{confirm}}$, then any protocol that satisfies $(\gamma, T_{\text{warmup}}, \xi)$ approximate-order-fairness must

also satisfy $(\gamma, T_{\text{warmup}})$ receive-order-fairness (for environments with a different network synchrony bound).

Theorem 2.10. *Consider any* $n, f \geq 1, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}}$. *Let* $\Delta_{\mathsf{int}} = (\mathsf{not\text{-}async}, \delta_{\mathsf{int}})$ *and* $\Delta_{\mathsf{ext}} = (\mathsf{not\text{-}async}, \delta_{\mathsf{ext}} \geq 1)$. *Also consider* $\gamma \leq 1$ *and* $\xi < T_{\text{confirm}}$. *If a protocol* Π *achieves consistency,* $(T_{\text{warmup}}, T_{\text{confirm}})$*-liveness, and* $(\gamma, T_{\text{warmup}}, \xi)$*-approximate-order-fairness. w.r.t. all* $(\mathcal{A}, \mathcal{Z})$ *that respect* $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$*-classical execution, then it also satisfies* $(\gamma, T_{\text{warmup}})$*-receive-order-fairness w.r.t. all* $(\mathcal{A}', \mathcal{Z}')$ *that respect* $(n, f, \Delta'_{\mathsf{int}}, \Delta'_{\mathsf{ext}})$*-classical execution where* $\Delta'_{\mathsf{int}} = (\mathsf{not\text{-}async}, \delta'_{\mathsf{int}} = \frac{\delta_{\mathsf{int}}}{\xi+1})$ *and* $\Delta'_{\mathsf{ext}} = (\mathsf{not\text{-}async}, \delta'_{\mathsf{ext}} = \frac{\delta_{\mathsf{ext}}}{\xi+1})$.

Proof. Deferred to Appendix A.1. □

Consequently, approximate order-fairness doesn't turn out to be very useful since it suffers from the same problems as the previously defined receive-order-fairness. Note that from Section 2.3.4, we can infer that approximate-order-fairness can be achieved when $\delta_{\mathsf{ext}} \leq \xi$. Still, since it only applies to non-asynchronous networks, we propose a second definition, batch-order-fairness, that performs much better since it provides a way to handle any cycles in transaction ordering and also applies to asynchronous networks. We note that our synchronous protocol (Section 3.3) also satisfies approximate-order-fairness for $\xi \geq \delta_{\mathsf{ext}}$.

For a given view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, we state the batch-order-fairness property below.

Definition 2.11 (Batch-Order-Fairness)**.** A view satisfies $(\gamma > \frac{1}{2}, T_{\text{warmup}})$-batch-order-fairness if the following holds: For any two transactions m and m', let η be the number of nodes that received both transactions between times T_{warmup}

and $|\mathsf{view}|$. If at least $\gamma\eta$ of those nodes received m before m' from $\mathcal{Z}$, then for all honest nodes i, i does not deliver m at a later index than it delivers m'.

A protocol Π satisfies $(\gamma, T_{\text{warmup}})$-batch-order-fairness w.r.t. $(\mathcal{A}, \mathcal{Z})$ if there is a negligible function $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, the above property is satisfied except with probability $\mathsf{negl}(\kappa)$ over a random choice of $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$.

In Chapter 3, we will show protocols that guarantee batch-order-fairness.

2.4 Comparing Order-Fairness to Related Works

While there is an extensive literature on consensus protocols, to the best of our knowledge, no previous work formally captures a notion of order-fairness like the one we introduce. We also note that the term "fairness" has been used widely in blockchain and cryptography literature, but for properties unrelated to ours.

Broadcast primitives. Byzantine broadcast, or the Byzantine Generals Problem [171], is the elementary broadcast primitive where a designated sender broadcasts a single value to a set of receiving nodes. In a Byzantine broadcast protocol with the key property of *consistency*, all honest receivers output the same value. Reliable broadcast is a continuous version of Byzantine broadcast where the sender broadcasts multiple values which must be eventually delivered by nodes if the sender is honest. Three orthogonal properties can be added onto reliable broadcast to give stronger notions. FIFO-ordering provides first-

in first-out ordering on the messages broadcast by an honest sender. We refer to such a protocol as (Single sender) FIFO Broadcast (also called OARcast for Ordered Authenticated Reliable Broadcast in [148]). Local-ordering (also called causal-ordering) ensures that if a node broadcasts a message m' after receiving some other message m, then m will be ordered before m'. The total-ordering property ensures that all honest nodes deliver messages broadcast potentially by different senders in the same order. This notion is usually called *atomic broadcast* [91], which is well-known to be equivalent to the consensus problem. Adding all three properties to reliable broadcast results in the notion of Causal FIFO Atomic Broadcast which still does not provide the *order-fairness* property that we are looking for. The main problem is none of the requirements consider a global notion of FIFO ordering based on multiple senders.

Our order-fairness property does enforce such a notion according to the following idea: If enough nodes broadcast a message m before another message m', then honest nodes will respect this ordering. Adding this property to atomic broadcast results in a *new broadcast* notion, which we call "Global FIFO Atomic Broadcast." Consequently, requiring order fairness along with standard consensus properties of consistency and liveness will be equivalent to this new notion of Global FIFO Atomic Broadcast.

We note that our setup is also slightly different than earlier notions. We assume that any message broadcast by an honest node is also eventually broadcast by all honest nodes. This allows us to redefine liveness in terms of being broadcast by enough nodes. This also means that identical messages broadcast by different nodes can now be delivered together as a single message. Global FIFO ordering is defined on the ordering of these messages. Note that it no longer

makes sense to talk about (single source) FIFO order or causal order as identical messages, potentially broadcast at different positions by different nodes, are now delivered as a single message.

Consensus protocols. Hundreds of Byzantine fault tolerant consensus protocols have been proposed over the years, with PBFT [64] being perhaps the most well known one. Multiple survey papers [24, 55] have recently aimed to systematize this vast literature. Many papers provide efficiency improvements while maintaining the basic leader-based structure of PBFT. That is, a *leader* or *primary* node is responsible for proposing the transactions in the current round. In such leader-based protocols ([183, 8, 17, 31, 84, 244, 243, 256, 3], just to name a few), the leader node can propose transactions in the order of its choosing. The leader is also capable of suppressing transactions, at least temporarily, until an honest node becomes the new leader. We highlight that in previously explored leader-based protocols, nodes do not know the ordering in which transactions were received by everyone else. This means that a leader's proposal can only be rejected by other nodes based on the validity of transactions rather than the fairness of their ordering. Order-fairness is thus not achieved in existing leader-based protocols.

Censorship resistance and blind ordering. Some protocols provide *transaction censorship resistance*, such that malicious nodes cannot censor specific transactions based on their content. For this, in protocols like [220, 53, 192, 14], transactions are encrypted and the contents are revealed only once their ordering is fixed. A similar objective can also be achieved through commit-and-reveal schemes [47, 5].

Separately, protocols in [167, 14, 172] rely on a reputation based system to detect unfair censorship. Censorship resistance is strictly weaker than the order-fairness we consider for three reasons. First, in practice, even if transaction data is temporarily encrypted, metadata such as a user identifier or a client IP address can be used to censor a particular transaction. Second, a malicious leader can still *blindly* reorder or censor transactions based on just their ciphertext. But perhaps more importantly, a malicious leader colluding with a user will know the ciphertext corresponding to the user's transaction and can thus unfairly order this transaction before others. An increasing number of attacks (e.g., displacement [109] and certain backrunning [182]) do not depend on transaction contents and therefore will not be stopped by these techniques.

Still, we note that broadly, privacy techniques (such as encrypted transactions) can provide complementary protection to our order-fairness notion. Privacy can protect against network-level manipulations in the *external* network, while order-fairness is geared towards the consensus layer.

Following our work, we also note that there has been renewed interest in designing more efficient protocols that support encrypted transactions [178] [ToDo: Add more references].

Random Ordering. It is important to distinguish a protocol that chooses a *random* ordering from a protocol that chooses a *fair* ordering. A *random* ordering protocol chooses a random ordering of transactions in the current pool, in an attempt to avoid any adversarial manipulation. On the other hand, a fair ordering protocol will guarantee that transactions which arrive earlier will be sequenced earlier by the network (instead of in a random order).

Even in a hypothetical perfect random ordering protocol where no metadata or side-channel information is leaked and where adversarial miners are provably forced to choose a random ordering of transactions in the mempool, there is still a 50% chance that an adversarial transaction gets ordered before an honest one even if the honest one was propagated earlier. In addition, the adversary can flood the network with many transactions to exponentially increase the probability that at least one is ordered before the honest transaction (e.g., clogging attacks [216]). We highlight that even if the adversary has no control over the ordering of transactions in the pool, it is able to implicitly influence the ordering by introducing its own adversarial transactions.

Random ordering is especially problematic if the adversary can use user transaction metadata to construct its own transaction, or if the adversarial transaction does not need to depend on the contents of any honest user's transaction (e.g., displacement attacks [109] where the adversary wants to be the first one through the door to for instance, buy an asset with only limited quantities available, or get the best price in an auction). This is not a hypothetical problem as such a flooding attack was performed during the BZRX token launch [182] to guarantee a better token buy price for the adversary. Therefore, a random ordering protocol is not sufficient to prevent order-manipulation attacks.

In contrast, our fair ordering protocols ensure that an adversary cannot tip the scales in its favor by creating many transactions.

Other uses of the word *fairness*. The term *fairness* has been used before in consensus literature for notions unrelated to ours. One popular use case relates to *fairness in block mining* in Proof-of-Work (PoW) blockchains, which intuitively requires that a node's mining rewards be proportional to its relative

computational power. That is, no node should be able to mine *selfishly* [114] to obtain more rewards than its fair share. This fairness notion is met by protocols in [204, 172, 177, 191, 2], among others.

Another related definition considers fairness in terms of the opportunities each node gets to append transactions to the ledger. This includes both fair leader election (in leader based protocols) and fair committee election (in hybrid consensus protocols). This definition is considered in [205, 172, 2, 164, 132]. We note that even if the leader election process is fair, the current leader still has the power to manipulate transaction ordering.

Fairness has also been used in the context of "fair exchange." Fair exchange protocols provide a way for mutually distrusting parties to exchange digital goods in a secure way. This notion is completely unrelated to ours but we mention it for completeness.

2.5 Other Order-Fairness Notions

Prior works. Two prior works mentioned "fair transaction ordering." Helix [14] only considers censorship resistance and fair committee election. It uses threshold encryption to choose a random set of pending transactions for inclusion in the current block. Hashgraph [23] intuitively considers our notion of receive-order fairness but provides no formal definitions. Moreover, it *fails to realize the impossibility* of this notion of fairness resulting from the Condorcet paradox [88]. As a result, we identify an elementary attack on the Hashgraph protocol that allows an adversarial node to control transaction ordering. We provide details in Section 2.5.1.

Subsequent works. Following the publication of this work, numerous other works [54, 170, 259, 81, 258, 163, 158, 196, 140, 218, 240, 85, 253, 73] have proposed alternate order-fairness definitions with varying tradeoffs and built protocols to realize them. While we make some comparisons to contemporary definitions as part of the content in e.g., Chapters 5 and 6, we emphasize that this thesis does not intend to provide an in-depth comparison. We also note that at the time of writing, no single order-fairness definition has been agreed upon by the community as the ideal desiderata.

2.5.1 Hashgraph Details

In the Hashgraph algorithm, each participant maintains a directed graph (called the hashgraph) of the transactions it has received from others. Participants *sync* their transactions to others by sending their local hashgraph to a randomly chosen participant at every round. The intuitive strategy of their consensus protocol is to ensure that the hashgraphs maintained by honest participants are consistent. When Alice receives a "sync" of the hashgraph from Bob, she adds all of Bob's new transactions (say including a transaction tx) and any of her own to a new event node N. She then sets the new node's parents to be the last node received from Bob, and her own last node. Alice includes a timestamp with the N which is considered to be Alice's receive-time for the transaction tx. Without going into too much detail, after N has been buried sufficiently deep in the graph, Alice considers a specific set of graph nodes in her hashgraph and computes the final timestamp for tx by taking the median of all the corresponding timestamps. Each participant ends up with the same final timestamp as they compute the median on the same set of event nodes. However, we highlight

that using the median to compute the final timestamp is the actual cause of unfairness since it is prone to adversarial manipulation. To see why, consider two users transactions tx_1 and tx_2 that are sent by honest users to all the protocol participants. Suppose that all nodes receive tx_1 before tx_2 and that the network adversary lets no "sync" attempts go through before everyone receives both tx_1 and tx_2. If the receive times for tx_1 and tx_2 are sufficiently intertwined, then even a single adversarial participant can cause the median timestamp for tx_1 to become larger than the median timestamp for tx_2 which breaks fair-ordering.

We acknowledge that carefully designing a different formal definition for fair ordering could allow the Hashgraph protocol to achieve a different notion of fairness, but we base our comparison on their informal notion of "first received, first output." In Section 3.2, we also show a simple concrete example of why median timestamp based ordering protocols do not work in general.

CHAPTER 3

THE Aequitas PROTOCOL FAMILY

This chapter constructs consensus protocols that also satisfy batch-order-fairness. The content in this chapter was initially published at CRYPTO 2020 as the work "Order-Fairness for Byzantine Consensus" [161].

3.1 Building Blocks

We start by describing some useful primitives that will form the foundation for designing our fair ordering consensus protocols. More specifically, we will utilize two primitives: (1) Set Byzantine Agreement (Set-BA); and (2) FIFO Broadcast (FIFO-BC). We introduce Set-BA in Section 3.1.1 and FIFO-BC in Section 3.1.2. We also show how to build Set-BA from standard Byzantine agreement and FIFO-BC from reliable broadcast.

Subroutines and composition. We follow the standard conventions to enable secure composition when considering multiple instantiations of the same protocol. Each instance of a protocol is spawned with a session identifier sid. We use $\Pi[\mathsf{sid}]$ to denote the instance of protocol Π with session id sid. Each protocol may take inputs from and return outputs to an environment. Note that this "environment" may be different for any subroutines called. For example, when a calling process p, forks an instance of a protocol Π, p is taken to be part of the environment for Π and handles its inputs and outputs.

3.1.1 Set Byzantine Agreement

Definitions. In a (poly) Set Byzantine Agreement protocol (Set-BA), participating nodes will try to agree on a set of values. At the start of the protocol, each node receives the identities of all participating nodes, the parameters n and f, the network parameters, as well as any other public parameters from $\mathcal{Z}$. At the start of the protocol, each node receives any public parameters from $\mathcal{Z}$. Each node i in the set $\mathcal{P}$ of participating nodes also receives a set $U_i \subseteq S$ as input from $\mathcal{Z}$. The set S is also known to all nodes and its size is polynomial in the parameters. At the end of the protocol, each honest node $j \in \mathcal{P}$ outputs a set of the agreed upon values O_j.

Definition 3.1 (Security of Set-BA)**.** A Set-BA protocol Π_{sba} satisfies agreement, inclusion validity, and exclusion validity w.r.t. $(\mathcal{A}, \mathcal{Z})$ if for all $\kappa \in \mathbb{N}$, the following properties hold except with negligible probability over the random choice of $\mathsf{view} \leftarrow\!\!\$\ \mathsf{EXEC}^{\Pi_{\text{sba}}}(\mathcal{A}, \mathcal{Z}, \kappa)$.

- **(Agreement)** If two honest nodes i and j output the sets O_i and O_j respectively, then $O_i = O_j$.
- **(Inclusion Validity)** If an element is in the input sets of all nodes, then it will also be in the output sets of all honest nodes. That is, if $c \in U_i$ for all $i \in \mathcal{P}$, then $c \in O_j$ for all honest j.
- **(Exclusion Validity)** If an element is not in any input set, then it is not in any honest output set. That is, if $c \notin U_i$ for all $i \in \mathcal{P}$, then $c \notin O_j$ for all honest j.

Comparison to Asynchronous Common Subset (ACS). A primitive related to Set-BA is the asynchronous common subset (ACS) problem from [53, 192] that can be used to build an asynchronous Byzantine fault tolerant system. Similar to our Set-BA primitive, each node in an ACS protocol is input a set U_i and all nodes agree on a common output set O. ACS guarantees that the common output O contains all the elements of the input sets of at least $n - 2f$ honest nodes. However, O can also contain elements that were proposed by only malicious nodes. On the other hand, a Set-BA protocol also needs to satisfy *exclusion validity* which along with the agreement and inclusion validity properties, guarantees that only honest proposals are included in output set. We prove this in Lemma 3.2. Note that the output set also need not include all elements from $n - 2f$ of the honest input sets.

As mentioned before, it is easy to prove that any Set-BA protocol satisfies the "honest proposal" property shown in Lemma 3.2.

Lemma 3.2. *Consider any set Byzantine agreement (Set-BA) protocol Π_{sba} that satisfies agreement, inclusion validity, and exclusion validity (w.r.t. $(\mathcal{A}, \mathcal{Z})$). Except for a negligible number of views, Π_{sba} also satisfies the following:*

- **(Honest Proposal)** *If an honest node outputs the set O, then for every $c \in O$, there exists $i \in \mathcal{P}$ such that i is honest and $c \in U_i$.*

Informally, this guarantees that all values in the agreed upon set must have been proposed by some honest node.

Proof. Proof deferred to Appendix B.1. □

Set Agreement from Binary Byzantine Agreement (BBA**).** We show how Set-BA can easily be realized from a BBA protocol. Recall that in a BBA protocol, each node i starts with an initial value $b_i \in \{0, 1\}$ and outputs a bit out_i when the protocol ends. The goal is for all honest players to output the same bit. A secure BBA protocol Π_{BBA} needs to satisfy two properties in all except a negligible number of executions —

- **(Agreement)** $out_i = out_j$ for all honest nodes i and j.
- **(Validity)** If all honest nodes start with the same initial value b, then $out_i = b$ for all honest nodes i.

Let Π_{BBA} be a BBA protocol that satisfies both agreement and validity. We can now construct a protocol Π_{sba} from the BBA protocol Π_{BBA} that satisfies the Set-BA security properties. Suppose that Π_{sba} needs to be instantiated with the session id sid. We now describe the protocol Π_{sba} for a node i:

1. For each $s \in S$, if $s \in U_i$, node i forks a new instance of $\Pi_{\text{BBA}}[(\mathsf{sid}, s)]$ with input 1; otherwise it forks an instance $\Pi_{\text{BBA}}[(\mathsf{sid}, s)]$ with input 0.
2. Collect the outputs of all Π_{BBA} instances. Let $out(s)$ denote the output of $\Pi_{\text{BBA}}[(\mathsf{sid}, s)]$. Construct the set $O = \{s \in S \mid out(s) = 1\}$ and output it.

Lemma 3.3. *If* Π_{BBA} *satisfies the* BBA *security properties for* $(\mathcal{A}, \mathcal{Z})$*, then* Π_{sba} *satisfies agreement, inclusion validity, and exclusion validity.*

Proof. Proof deferred to Appendix B.1. □

For our purpose, an equivalent way to view Set-BA is a combination of individual Byzantine agreement for every possible input element in S. Our pro-

tocols consider $S = \{1, \ldots, n\}$ and we will use the Set-BA primitive to agree on the which local node orderings to consider to finalize the order of a given transaction.

Other Properties. To analyze other useful characteristics of a Set-BA protocol, we define two additional properties, liveness and α-validity. Liveness describes how long it takes for nodes to reach agreement while α-validity can be used to determine how easy it is for an adversary to make honest nodes agree on a non-majority value.

Formally, we say that a protocol Π_{sba} satisfies $T^{\text{sba}}_{\text{confirm}}$-liveness (respectively α_{sba}-validity) if the properties as described below are satisfied except for a negligible number of executions.

- **($T^{\text{sba}}_{\text{confirm}}$-Liveness)** All honest nodes output in at most $T^{\text{sba}}_{\text{confirm}}$ rounds after all honest nodes have input their starting value.

  When the network is asynchronous, we define liveness in the same way as for state machine replication.

- **(α_{bba}-Validity)** If c is present in the initial sets of at least α_{bba} fraction of all nodes, then $c \in O_i$ for all honest nodes i.

$T^{\text{sba}}_{\text{confirm}}$ is a polynomial in κ, n, f, the network delay bound in Δ_{int}, and the actual internal network delay.

3.1.2 FIFO Broadcast

Single source FIFO (first in, first out) broadcast (also called Ordered Authenticated Reliable broadcast or OARcast in [148]) is a broadcast primitive in which all honest nodes in the protocol need to deliver messages in the same order as they were broadcast by the sender. In one instantiation of a FIFO broadcast protocol, we consider a single designated sender who broadcasts a sequence of messages to all other nodes. If the sender is honest, each honest node must deliver the messages in the same order as they were broadcast. If the sender is dishonest, all honest nodes must deliver messages in the same order as each other; except now, this order may may be different than the one broadcast by the sender. When composing several FIFO broadcast primitives together with different senders, FIFO order is maintained for each individual sender but different honest nodes may deliver messages from different senders in different orders.

Definitions. At the start of the FIFO Broadcast ($\mathsf{FIFO\text{-}BC}$) protocol, each node receives the appropriate public parameters from the environment. At any time, the designated sender may also receive as input a message m from the environment. At any time, nodes can choose to deliver messages.

Definition 3.4 (Security of ($\mathsf{FIFO\text{-}BC}$))**.** A $\mathsf{FIFO\text{-}BC}$ protocol Π_{fifocast} satisfies liveness, agreement, and FIFO-order w.r.t. $(\mathcal{A}, \mathcal{Z})$ if for all $\kappa \in \mathbb{N}$, the following properties hold except with negligible probability over the random choice of $\mathsf{view} \leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{fifocast}}}(\mathcal{A}, \mathcal{Z}, \kappa)$.

- **($(T^{\text{fifocast}}_{\text{warmup}}, T^{\text{fifocast}}_{\text{confirm}})$-Liveness)** If the sender is honest and receives a message m as input in round $r > T^{\text{fifocast}}_{\text{warmup}}$, or if an honest node delivers m in round

$r > T^{\text{fifocast}}_{\text{warmup}}$, then all honest nodes will have delivered m by round $r + T^{\text{fifocast}}_{\text{confirm}}$.

Eventual liveness in asynchronous networks is defined in the same way as for state machine replication.

- **(Agreement)** If an honest node delivers a message m before m', then no honest node delivers m' unless it has already delivered m.
- **(FIFO-Order)** If the sender is honest and is input a message m before m', then no honest node delivers m' unless it has already delivered m.

$T^{\text{fifocast}}_{\text{confirm}}$ is a polynomial in κ, n, f, the network delay bound in Δ_{int}, and the actual internal network delay.

Notation. Let $\Pi_{\text{fifocast}}[(\mathsf{sid}, j)]$ denote the instance of the protocol Π_{fifocast} where node j is the designated sender. In a consensus protocol that invokes $\Pi_{\text{fifocast}}[(\mathsf{sid}, j)]$, we assume that each node i keeps track of the messages delivered (i.e. messages broadcast by node j) in a local log $\mathsf{Log}_i^{(\mathsf{sid},j)}$. This represents node i's view of broadcasts from node j in the session sid. When the session id is clear from context, we may also write the local log simply as Log_i^j.

Two local logs Log and Log' are called "equal until tx", denoted by $\approx_{\mathsf{tx}}$, if they are equivalent until the occurrence of tx. $\mathsf{Log}[p]$ denotes the p^{th} element in Log. $\mathsf{Log}(m)$ denotes the number p such that $\mathsf{Log}[p]$ contains m. Consequently, $\mathsf{Log}(m) < \mathsf{Log}(m')$ signifies that m appears before m' in Log.

FIFO-BC from Reliable Broadcast. Reliable broadcast is a basic broadcast primitive where a designated sender broadcasts messages to a set of nodes.

Honest nodes will only deliver those messages that were broadcast, and will eventually deliver all messages broadcast by an honest sender. Reliable broadcast can be considered a "continuous" version of single shot Byzantine broadcast or the Byzantine generals problem [171]. Ho et al. [148] show how FIFO broadcast can be achieved using reliable broadcast even in asynchronous networks. The intuition is simple: sequence numbers are added to the messages broadcast by the sender in a reliable broadcast protocol. An honest node does not deliver a message with sequence number k until it has delivered a message with sequence number $k-1$. We refer the reader to [148] for the detailed construction.

3.2 General Design Overview

We provide a general overview of our Aequitas protocols in this section. In the next two sections, we will dive deeper into the actual Aequitas constructions. Specifically, we provide four concrete protocols: $\Pi^{\mathsf{sync,lead}}_{\mathsf{Aequitas}}$, $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$, $\Pi^{\mathsf{async,lead}}_{\mathsf{Aequitas}}$ and $\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$. Sections 3.3 and 3.4 describe the leaderless synchronous and asynchronous protocol designs respectively. The leader-based protocols are easier modifications to existing consensus protocols and we use Section 3.5 to briefly discuss them.

- $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ is a leaderless protocol that provides consistency, weak-liveness, and batch-order-fairness in the completely synchronous setting.
- $\Pi^{\mathsf{sync,lead}}_{\mathsf{Aequitas}}$ is a leader-based protocol that provides consistency, weak- liveness, and batch-order-fairness in the completely synchronous setting.

- $\Pi_{\text{Aequitas}}^{\text{async,nolead}}$ is a leaderless protocol that provides consistency, eventual-weak-liveness, and batch-order-fairness in any setting.
- $\Pi_{\text{Aequitas}}^{\text{async,lead}}$ is a leader-based protocol that provides consistency, eventual-weak-liveness, and batch-order-fairness in any setting.

Construction overview. Aequitas protocols utilize the FIFO-broadcast (FIFO-BC) and the set byzantine agreement (Set-BA) primitives described in Section 3.1 in a black-box way to provide order-fairness. At a high level, Aequitas protocols function in three steps: First, a node uses a FIFO-BC protocol Π_{fifocast} to send all other nodes the transactions it has received from users. Recall that in FIFO-BC, nodes deliver messages in the same order as broadcast by an honest sender. When a node delivers a message received from another node, it gets added to its local log. To elaborate, broadcasts from node j as delivered by node i are tracked in the local log Log_i^j. Next, all nodes seek to agree on the content of these local logs so as to order the transaction tx in question. This is done using a Set-BA protocol Π_{sba}. At this point, intuitively, all honest nodes have agreed on anything that will be used to compute the ordering for tx. To decide on the final ordering for tx, we provide two options for the finalization step — a leader-based one and a leaderless one.

For the finalization step in the leader-based protocol, a designated leader proposes an extension to the current chain. Since other nodes have all the relevant transaction orderings from the stages before, they can verify that the leader's proposal does not break order-fairness. If the leader's proposal is valid, nodes can deliver the proposed transactions by extending their LOG output to $\mathcal{Z}$. An important difference exists between such a leader-based protocol and prior leader-based protocols: In earlier protocols, a leader could propose any

ordering of its choice that would be accepted by other nodes. On the other hand, in our leader-based protocol, a malicious leader can mess with the transaction orderings only in a way that does not break the order-fairness property. For instance, if a transaction tx_1 was received before tx_2 by all nodes, a malicious proposal that puts tx_2 before tx_1 will be rejected by all the other nodes.

We propose another finalization that is leaderless and requires no further communication between nodes. It also provides consistency, batch-order-fairness and weak-liveness (from Section 2.2.3). Recall that "weak" denotes that liveness depends on transactions continuing to be input into the system.

We elaborate on the three major stages of our Aequitas protocols below:

- **Stage I: Gossip / Broadcast**. Each node FIFO-broadcasts transactions as they are received as input from the environment. When a node i receives a set of transactions txs from $\mathcal{Z}$, it sends txs as input to the protocol $\Pi_{\text{fifocast}}[(\mathsf{sid}, i)]$ with i as the designated sender. Note that all broadcasts can be sent in the same session sid. Different session ids need to be used only when considering composition of several protocols in the system.

  In parallel to broadcasting transactions, a node also receives and processes broadcasts from other nodes. For a node i, broadcasts sent by node j are appended to a local log Log_i^j when they get delivered to i by $\Pi_{\text{fifocast}}[(\mathsf{sid}, j)]$. Intuitively, Log_i^j denotes node i's view of how transactions were received by node j.

- **Stage II: Agreement on local logs**. To determine the ordering for a partic-

ular transaction tx, a node i waits until it has received tx from sufficiently many other nodes. In other words, node i waits until there are sufficiently many k such that its local log Log_i^k contains tx. When both the external and internal networks are synchronous, this can alternatively be achieved by waiting for enough *time*. The properties of FIFO-BC guarantee that if two honest nodes i and j have local logs Log_i^k and Log_j^k respectively that both contain tx, then $\mathsf{Log}_i^k \approx_{\mathsf{tx}} \mathsf{Log}_j^k$. We state this fact as Lemma 3.5. Recall that $\mathsf{Log}_i^k \approx_{\mathsf{tx}} \mathsf{Log}_j^k$ holds when Log_i^k and Log_j^k are identical until tx occurs.

Now, the next step is for all nodes to agree on which local logs to use to determine the ordering for tx. For a node i, let U_i^{tx} denote the set of nodes k such that Log_i^k contains tx. Node i starts an instance of the protocol $\Pi_{\mathsf{sba}}[(\mathsf{sid}, \mathsf{tx})]$ and provides it the input U_i^{tx}. Upon the completion of the Set-BA protocol, all honest nodes receive the same set L^{tx}. Intuitively, Set-BA is used to agree which nodes' orderings should be used to determine the final ordering for transaction tx. Recall that Lemma 3.2 guarantees that if $k \in L^{\mathsf{tx}}$, then there is some honest node j such that $\mathsf{tx} \in \mathsf{Log}_j^k$. This, along with the liveness property for FIFO-BC ensures that all honest nodes will eventually receive tx broadcast by node $k \in L^{\mathsf{tx}}$ (even if k is malicious).

Finally, we note that at the end of the agreement phase, every honest node has agreed on a set of nodes L^{tx} whose transaction orderings should be used to determine the final ordering for the transaction tx in consideration. We say that a node i has received the agreed logs for tx if for all $k \in L^{\mathsf{tx}}$, it holds that $\mathsf{tx} \in \mathsf{Log}_i^k$.

- **Stage III: Finalization.** To decide on the final ordering for a transaction

tx, we provide two options for the finalization step: a leader-based one and a leaderless one. For both the leader-based and leaderless finalizations, nodes first build a graph that represents any ordering dependencies between transactions. Specifically, a node i maintains a directed graph G_i, where vertices represent transactions and edges represent ordering dependencies. We refer to G_i as the "dependency graph" or the "waiting graph" maintained by i. After the agreement stage for tx is completed, the protocol now uses the local logs to see if some other transaction might have come before. If there is another transaction tx′ that appears before tx in sufficiently many local logs (e.g., $n - f$ times), then i adds an edge from tx′ to tx in G_i. Intuitively, an edge $(a, b) \in G_i$ denotes that the finalization stage for b is "waiting" for a to be delivered. Since the same L^{tx} is used by all honest nodes, if an edge (a, b) exists in G_i, then it will at some point exist in G_j, when nodes i and j are both honest. However, we note that G_i is neither guaranteed to be complete nor acyclic. Two vertices in G_i that might never have an edge between them. Moreover, the Condorcet paradox can still create cycles in G_i. To break ties between transactions without an edge, we use the following two techniques.

– **Finalization via leader-based proposal.** $\Pi^{\mathsf{sync,lead}}_{\mathsf{Aequitas}}$ and $\Pi^{\mathsf{async,lead}}_{\mathsf{Aequitas}}$ both use a leader-based approach to finalize transactions in the graph. For this, any leader-based consensus protocol can be run along with the gossip and agreement stages above. When a designated leader proposes and broadcasts a new block, instead of just checking the syntactical validity of transactions, each node i checks that the proposal does not conflict with any required order-fairness in the graph G_i. That is, node i checks that for any transaction tx in the proposed

block, if $(\mathsf{tx}', \mathsf{tx})$ is in G_i, then either tx' has already been delivered or tx' is also in the current proposed block.

Abstractly, we allow the leader node to choose the transaction ordering but only as long as order-fairness is still satisfied. For transactions among which there is no clear winner, the leader may choose any ordering.

– **Finalization via local computation.** $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ and $\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$ both use a leaderless approach to finalize transactions in the graph and require no further communication. At a high level, to order transactions tx_1 and tx_2 between whom there in no edge in G_i, the protocol will wait until tx_1 and tx_2 have a common descendant, with the final ordering being based on which transaction vertex has the most descendants. We prove that any other graph vertex that is a descendant of only one of tx_1 and tx_2 is present in G_i when node i makes the decision for ordering tx_1 and tx_2. This will ensure that all honest nodes will order tx_1 and tx_2 the same way.

We highlight that the above description of the finalization stage is a simplified one. As described, it is not sufficient to avoid the Condorcet paradox. Furthermore, adversarial transactions could result in a node waiting for unbounded periods of time. The actual technique to get around these obstacles is quite nuanced and we dedicate Section 3.2.1 to its details.

Lemma 3.5. *If two honest nodes i and j have local logs Log_i^k and Log_j^k respectively where k is any other node such that both logs contain a transaction* tx*, then $\mathsf{Log}_i^k \approx_{\mathsf{tx}} \mathsf{Log}_j^k$.*

Proof. This result follows directly from the agreement property of FIFO-BC. □

Before diving into the details of the finalization step, we take a step back to understand why it turns out to be quite non-trivial. We look at a simple strawman protocol based on transaction timestamping that looks intuitive and analyze why it does not work.

The problem with timestamp-based ordering. Consider a simple synchronous protocol $\Pi_{\text{timestamp}}$ that works as follows:

1. When an honest node i receives a transaction tx from $\mathcal{Z}$ in round t, it assigns tx the timestamp t and broadcasts (tx, t) to all other nodes.
2. Upon waiting for $\delta_{\text{ext}} + T_{\text{confirm}}$ rounds where δ_{ext} is the network delay bound for the external network and T_{confirm} is the liveness polynomial for the broadcast primitive, nodes reach agreement on the set of timestamps T to use to calculate the final timestamp for tx.
3. Each node calculates the final timestamp for tx as the median of all the timestamps in T. We represent this final timestamp by $\mathsf{final}(\text{tx})$.

Notice how the first two steps almost perfectly resemble the gossip and agreement stages. The finalization (third) step is also surprisingly simple, but unfortunately can lead to easy manipulation of final timestamps by a single adversary. To see why, consider 5 nodes, A, B, C, D and E, where E is malicious and two transactions, tx_1 and tx_2. tx_1 is received by nodes $A, \ldots, E$ at rounds $1, 1, 4, 4, 2$ while tx_2 is received by the nodes at rounds $2, 2, 5, 5, 3$. Now, all nodes have received tx_1 before tx_2 and consequently, $\mathsf{final}(\text{tx}_1) < \mathsf{final}(\text{tx}_2)$ should hold. However, notice how E can invert the ordering of the final timestamps simply by switching around its own timestamps for tx_1 and tx_2. E can

make $\mathsf{final}(\mathsf{tx}_1) = 3$ and $\mathsf{final}(\mathsf{tx}_2) = 2$, i.e., a final timestamp of 3 for tx_1 (median of (1, 1, 3, 4, 4)) and 2 for tx_2 (median of (2, 2, 2, 5, 5)), and thus an unfair ordering.

3.2.1 The Finalization Stage

We describe the general theme of the finalization stage here.

Ordering two transactions. For a pair of transactions tx and tx′, how does a node i choose which one to deliver first? Suppose that the agreement phases for tx and tx′ result in the outputs L^{tx} and $L^{\mathsf{tx}'}$. Define $l_{(\mathsf{tx},\mathsf{tx}')}$ as below.

$$l_{(\mathsf{tx},\mathsf{tx}')} = \left|\left\{k \in L^{\mathsf{tx}} \cup L^{\mathsf{tx}'} \mid \mathsf{Log}_i^k(\mathsf{tx}) \leq \mathsf{Log}_i^k(\mathsf{tx}')\right\}\right|$$

$l_{(\mathsf{tx},\mathsf{tx}')}$ denotes the number of logs Log_i^k where tx was ordered at or before tx′. Now, if $l_{(\mathsf{tx},\mathsf{tx}')}$ is "small," it means that a large number of nodes have received tx′ before tx. This means that the finalization stage for tx should wait until tx′ has been delivered. This provides a partial ordering between any two transactions. We defer the details to when we describe the actual Aequitas constructions.

Additional notation. Let $\mathsf{tx} \lhd_i \mathsf{tx}'$ represent that i is waiting to deliver tx′ before proceeding with the finalization phase for tx. Lemma 3.6 shows that $l_{(\mathsf{tx},\mathsf{tx}')}$ and $l_{(\mathsf{tx}',\mathsf{tx})}$ cannot both be "small". Consequently, both tx and tx′ will not wait for each other or equivalently, at most one of $\mathsf{tx} \lhd_i \mathsf{tx}'$ and $\mathsf{tx}' \lhd_i \mathsf{tx}$ will be true.

Lemma 3.6. $l_{(\mathsf{tx},\mathsf{tx}')} + l_{(\mathsf{tx}',\mathsf{tx})} \geq \left|L^{\mathsf{tx}} \cup L^{\mathsf{tx}'}\right|$

Proof. Let $X = L^{\mathsf{tx}} \cup L^{\mathsf{tx}'}$. For any $k \in X$, at least one of $\mathsf{Log}_i^k(\mathsf{tx}) \leq \mathsf{Log}_i^k(\mathsf{tx}')$ and $\mathsf{Log}_i^k(\mathsf{tx}') \leq \mathsf{Log}_i^k(\mathsf{tx})$ is true. k is therefore counted in either $l_{(\mathsf{tx},\mathsf{tx}')}$ or $l_{(\mathsf{tx}',\mathsf{tx})}$ which proves the required result. □

Adversarial transactions. The calculation of $l_{(\mathsf{tx},\mathsf{tx}')}$ needs to wait for the agreement phases of both tx and tx′ to finish. Now, if an adversarial node FIFO-broadcasts a transaction $\mathsf{tx}_{\mathsf{fake}}$ claiming it to be a real user transaction, then the ordering between $\mathsf{tx}_{\mathsf{fake}}$ and a real transaction tx cannot be calculated since the agreement phase for $\mathsf{tx}_{\mathsf{fake}}$ will never finish. So that this does not happen, the protocol needs to ensure that at least one honest node has received $\mathsf{tx}_{\mathsf{fake}}$ before tx (from $\mathcal{Z}$). For example, in the synchronous protocol, this is done by checking that a transaction tx′ is added to the graph only when there is another transaction tx that has finished its agreement stage and tx′ is present in at least $|L^{\mathsf{tx}}| - (n - f) + 1$ among the local logs in L^{tx}. Note that the agreement stage will only finish for honest transactions.

Non-transitive waiting. The Condorcet paradox can still cause non-transitive waiting. It is still possible to have transactions $\mathsf{tx}_1, \mathsf{tx}_2$, and tx_3 such that $\mathsf{tx}_1 \lhd \mathsf{tx}_2$; $\mathsf{tx}_2 \lhd \mathsf{tx}_3$; and $\mathsf{tx}_3 \lhd \mathsf{tx}_1$. The way we get around this is by delivering such transactions at the same time—by placing them in the same batch.

Graph based approach. Instead of a separate thread waiting for the resolution of each transaction, representing the "waiting" between transactions as a graph provides a nice way to modularize the protocol. Suppose that each node i maintains a directed graph $G_i = (G_i.V, G_i.E)$ where $G_i.V$ denotes the set of vertices and $G_i.E$ denotes the set of edges in G_i. Each vertex represents a transaction

and an edge from y to x (equiv. $(y, x) \in G_i.E$) represents that x is waiting on y i.e. $x \lhd_i y$. When the agreement phase for a transaction tx completes, i does the following:

- Add tx to the graph G_i if it does not already exist.
- For all transactions tx′ such tx $\lhd_i$ tx′, first, if tx′ does not exist in the graph, add a new vertex. Then, add the edge (tx', tx) to G_i.

As mentioned before, G_i may not be acyclic. In order to deal with the Condorcet paradox, we consider the *strongly connected components* of G_i. Recall that a subgraph G' of a directed graph G is called strongly connected if every vertex in G' can reach every other vertex in G'. A strongly connected component is a maximal strongly connected subgraph.

Intuitively, all transactions in a strongly connected component will be delivered in the same batch. A cycle that exists in G_i (due to non-transitivity of transactions) will be entirely contained in the same strongly connected component. On the other hand, if a transaction does not need to wait on any other one, then it will be in a strongly connected component by itself. We can collapse G_i into a new graph G_i^* where each strongly connected component is represented as a single vertex. G_i^* is also called the *condensation* of G_i. Each vertex in G_i^* will now denote a set of transactions. We note that G_i^* will now be acyclic.

Graph Notation. Since a vertex in G_i contains a single transaction, we may use a transaction and its corresponding vertex interchangeably when referring to the vertex in G_i. Let $\mathsf{TXS}_i(v)$ be the set of transactions for a vertex $v \in G_i^*.V$. Let $\mathsf{SCC}_i(v)$ denote the strongly connected component of G_i that contains the

vertex v. $\mathsf{SCC}_i(v)$ also denotes the corresponding vertex in the condensation graph G_i^*.

Ordering incomparable vertices in G_i^* and breaking ties. As mentioned before, not all pairs of vertices in G_i^* are connected by an edge. This only gives a partial ordering for delivering transactions. We still need a way to totally order vertices in G_i^*. In the leader-based version of the finalization step, we delegate this responsibility to the leader node. We elaborate on the technique used in the synchronous leaderless protocol in Section 3.3 and the asynchronous leaderless protocol in Section 3.4.

Delivering a transaction. Recall that a transaction enters the *finalization* stage when it has completed the agreement stage, while it is *delivered* when it gets output to $\mathcal{Z}$ as part of the LOG. For the leaderless protocols, the set of transactions $\mathsf{TXS}_i(v)$ corresponding to the vertex $v \in G_i^*.V$ can be delivered in the LOG output to $\mathcal{Z}$ when it is not waiting for any other transaction and is preferred over any other transaction that it is incomparable with in the graph. For this, care must be taken to ensure that the set of transactions that tx is incomparable with is the same when all honest nodes are deciding to deliver tx, which we defer to the actual protocol descriptions in Sections 3.3 and 3.4.

3.2.2 Adding Order-Fairness to Any Consensus Protocol

One of the upshots of our $\mathsf{Aequitas}$ constructions is that they provide a generic compiler that allows any standard consensus protocol to be converted into one

that provides order-fairness. Aequitas protocols only rely on reliable broadcast and Byzantine agreement, both of which can be realized by any existing consensus protocol.

3.3 The Synchronous Aequitas protocol

We describe $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$, the leaderless Aequitas protocol for the completely synchronous setting. By "complete synchrony," we mean that both the external and internal networks are synchronous. For this section, we assume that $(\mathcal{A}, \mathcal{Z})$ respects $\Delta_{\mathsf{ext}} = (\mathsf{full}, \delta_{\mathsf{ext}})$ ext-synchrony and $\Delta_{\mathsf{int}} = (\mathsf{full}, \delta_{\mathsf{int}})$ int-synchrony.

To build the $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ protocol, we assume a secure FIFO-BC protocol Π_{fifocast} (from Definition 3.4) and a Set-BA secure protocol Π_{sba} (from Definition 3.1) that work for any $(\mathcal{A}, \mathcal{Z})$ that respect $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$-classical execution. Let $(T^{\text{fifocast}}_{\text{warmup}}, T^{\text{fifocast}}_{\text{confirm}})$ and $T^{\mathsf{Set\text{-}BA}}_{\text{confirm}}$ denote the liveness parameters for Π_{fifocast} and Π_{sba} respectively. We note that any bound for the number of corruptions f will be at least as restrictive as bounds required by Π_{fifocast} and Π_{sba}.

3.3.1 Protocol Description

The $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ protocol follows much of the same general techniques from Section 3.2. The gossip and agreement stage take place exactly as described there. In the gossip stage, a node i forks an instance of $\Pi_{\text{fifocast}}[(\mathsf{sid}, i)]$ and uses it to broadcast transactions as they are received from $\mathcal{Z}$. After broadcasting a transaction tx, it waits until the broadcasts from all honest nodes would have arrived. Let U_i^{tx} denote the set of nodes k such that $\text{tx} \in \mathsf{Log}_i^k$. Note that all honest nodes

are present in U_i^{tx}. In the agreement stage, i forks an instance of $\Pi_{\mathsf{sba}}[(\mathsf{sid}, \mathsf{tx})]$ to agree on a set L^{tx} indicating the nodes whose logs to use to order tx.

For the finalization stage, we now present the remaining details that were deferred from Section 3.2.1. Please refer to Section 3.2 for any notation.

Building the "waiting" graph G_i. Recall that each node i builds a graph G_i where vertices are transactions and edges denote ordering dependencies between transactions. For two transactions tx and tx′, an edge $(\mathsf{tx}', \mathsf{tx})$ is added to G_i if $l_{(\mathsf{tx},\mathsf{tx}')} \leq \left|L^{\mathsf{tx}} \cup L^{\mathsf{tx}'}\right| - \gamma n + f$. Each node i also maintains the condensation graph G_i^* where each strongly connected component in G_i is condensed to a single vertex.

Ordering incomparable vertices in G_i^*. Suppose that v and v' are two vertices in G_i^* that are are currently not comparable i.e. they do not have an edge between them. To determine which vertex to deliver first, we wait until they have a common descendant, after which we order based on number of descendants. We note that once a common descendant arrives, any other transaction that arrives will also be a descendant of both v and v'. In other words, the vertex with the higher number of descendants will become fixed allowing for a consistent ordering across protocol nodes. Lemma 3.7 shows a helpful result on when vertices can be "incomparable."

A subtle point to note here is that the common descendant itself can cause v and v' to be combined into the same strongly connected component if it creates a cycle containing them. This is precisely why our protocol achieves weak-liveness, where we achieve liveness, if a transaction arrives late enough that

it cannot create a cycle with transactions in v and v'. Effectively, we need to wait for a transaction to arrive at a sufficiently later time in order to "flush out" earlier transactions.

Lemma 3.7. *Let v_1 and v_2 be two vertices in G_i^* that do not have an edge between them. Let r_{first} denote the time when any transaction in $\mathsf{TXS}(v_1)$ was first received by a node. Let r_{last} denote the time when any transaction in $\mathsf{TXS}(v_2)$ was last received by a node. Then $r_{\mathit{last}} - r_{\mathit{first}} \leq 2\delta_{\mathsf{ext}}$.*

Proof. Deferred to Appendix B.2. □

Breaking ties. We use an a priori known ordering relation to break any ties that arise (e.g., two vertices with equal number of descendants). In particular, suppose that Ord is a binary relation on $2^{\mathcal{T}} \times 2^{\mathcal{T}}$ that is known a priori to all nodes. $2^{\mathcal{T}}$ represents the power set of $\mathcal{T}$. The relation is defined on sets of transactions (rather than individual transactions only) since we may deliver several transactions at once. We assume that Ord is supplied to all nodes on initialization by $\mathcal{Z}$. We will use this function to deterministically break ties between two sets of transactions when neither should clearly come before the other. For two sets S_1 and S_2, $(S_1, S_2) \in \mathsf{Ord}$ implies that all nodes agree S_1 should come before S_2 if there is no clear winner. Ord can also be used to order transactions in the same block. In general, the Ord relation only needs to satisfy two properties:

- $\forall (a, b) \in 2^{\mathcal{T}} \times 2^{\mathcal{T}}; a \neq b$, exactly one of (a, b) and (b, a) is in Ord.
- $\forall a, b, c \in 2^{\mathcal{T}}$, if $(a, b) \in \mathsf{Ord}$ and $(b, c) \in \mathsf{Ord}$ then $(a, c) \in \mathsf{Ord}$.

We note that Ord can be defined using a simple alphabetical or ascending order.

Delivering transactions. The transactions $\mathsf{TXS}_i(v)$ of a vertex v in G_i^* can be delivered when:

- v is a source vertex i.e. it has no incoming edge. This ensures that v is not waiting on any other transaction to be delivered first.
- $3\delta_{\mathsf{ext}}$ rounds have passed since v was added to the graph. Waiting for $2\delta_{\mathsf{ext}}$ rounds ensures that any other vertex v' that v is incomparable to, is also present in the graph. Waiting for another δ_{ext} rounds ensures that any vertex that is a descendant of only one of v such a v' is also present in the graph.
- For any other source vertex v', v has a common descendant with v' and either has more descendants or has an equal number of descendants and $(\mathsf{TXS}_i(v), \mathsf{TXS}_i(v')) \in \mathsf{Ord}$ holds. This ensures that every node will order v before v'.

Bound on f. Suppose that $(\gamma, \cdot)$ order-fairness needs to be realized. This implies that if γn nodes receive transactions in a particular order, it must be reflected in the final ordering. Since f nodes can be adversarial, the output must be the same even if $\gamma n - f$ of those orderings are seen. Now, as we don't want a bi-directed edge to be added to G_i (this can lead to an unbounded length cycle), $\gamma n - f > \frac{n}{2}$ must hold. Equivalently, $n > \frac{2f}{2\gamma - 1}$. For $\gamma = 1$ batch-order-fairness, we require an honest majority.

Communication Complexity. Let s be the size (in bits) of a transaction. Let $\mathcal{B}(\lambda)$ be the communication complexity of an optimal λ-bit Byzantine agreement protocol. Then, reliable broadcast for one sender can be instantiated with

$\mathcal{O}(\mathcal{B}(s))$ communication for each s-bit transaction tx. Consequently, for each transaction, $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ has communication complexity $\mathcal{O}(n\mathcal{B}(s) + n\mathcal{B}(1))$.

3.3.2 Protocol Pseudocode

Initialization. At the start of the protocol, we assume that i receives the identities of other protocol nodes, n, f, the maximum network delays δ_{int}, δ_{ext}, and the binary relation Ord. A FIFO-BC protocol Π_{fifocast} and a Set-BA protocol Π_{sba} have also been agreed upon a priori. Let $T^{\text{fifocast}}_{\text{confirm}}$ and $T^{\text{sba}}_{\text{confirm}}$ represent the liveness bounds for Π_{fifocast} and Π_{sba} respectively. Now, for each $j \in \mathcal{N}$, i initializes $\mathsf{Log}^j_i \leftarrow []$. It also initializes an empty graph G_i and a final output log LOG_i.

• At the start of a round r, when i receives a set of transactions txs from $\mathcal{Z}$, it does the following:

1. **(Gossip)**
   (a) Fork an instance of $\Pi_{\text{fifocast}}[(\mathsf{sid}, i)]$ with i as the sender, if it does not already exist.
   (b) Send txs as input to $\Pi_{\text{fifocast}}[(\mathsf{sid}, i)]$.
   (c) Record $(\mathsf{sid}, \mathsf{txs}, \textsf{gossip-end}, r + \delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}})$
2. **(Agreement)**
   (a) Check if there is any previously recorded tuple $(\mathsf{sid}, \textsf{gossip-end}, \mathsf{txs}', r')$ such that $r = r'$.

(b) For such a tuple for txs', for each $\mathsf{tx} \in \mathsf{txs}'$, fork an instance of $\Pi_{\mathsf{sba}}[(\mathsf{sid}, \mathsf{tx})]$ and provide it the input U_i^{tx}.

(c) Record $(\mathsf{sid}, \textsf{agreement-end}, \mathsf{tx}, r + T^{\mathsf{sba}}_{\mathrm{confirm}})$ for each $\mathsf{tx} \in \mathsf{txs}'$.

3. **(Build Graph)**

(a) Check if there is any previously recorded tuple $(\mathsf{sid}, \textsf{agreement-end}, \mathsf{tx}, r')$ such that $r = r'$.

(b) For such a tuple for tx, first add a vertex denoted by tx to G_i if it does not already exist. Now, for any other transaction tx' seen so far that has not yet been delivered,

i. Let $u = \left|\left\{k \in L^{\mathsf{tx}} \mid \mathsf{tx}' \in \mathsf{Log}_i^k\right\}\right|$.

ii. If $u \geq |L^{\mathsf{tx}}| - (n - f) + 1$, compute $l_{(\mathsf{tx},\mathsf{tx}')}$ as per Section 3.2.1.

iii. If $l_{(\mathsf{tx},\mathsf{tx}')} \leq \left|L^{\mathsf{tx}} \cup L^{\mathsf{tx}'}\right| - \gamma n + f$, then record $\mathsf{tx} \lhd \mathsf{tx}'$. Add an edge $(\mathsf{tx}', \mathsf{tx})$ to G_i if it does not already exist.

(c) Record $(\mathsf{sid}, \textsf{graph-end}, \mathsf{tx}, r + 3\delta_{\mathsf{ext}})$ for tx

4. **(Finalization)**

(a) Compute the *condensation* graph G_i^* of G_i by collapsing each strongly connected component into a single vertex.

(b) Let V_{source} be the set of vertices in G_i^* where $v \in V_{\mathsf{source}}$ if it satisfies:

- All transactions in $\mathsf{TXS}(v)$ have been received.
- v is a source vertex in G_i^*. That is, v has no incoming edges.

(c) Let $V_{\mathsf{finalize}} \subseteq V_{\mathsf{source}}$ be the set of vertices v that also satisfy:

- For all $\mathsf{tx}^* \in \mathsf{TXS}(v)$, there is any previously recorded tuple $(\mathsf{sid}, \textsf{graph-end}, \mathsf{tx}^*, r')$ with $r \geq r'$

(d) For $v \in V_{\mathsf{source}}$, let $\mathsf{Desc}(v)$ denote the descendants of v in G_i^*. Let $\mathsf{nDesc}(v) = |\mathsf{Desc}(v)|$ i.e. the number of descendants.

(e) For $v \in V_{\mathsf{finalize}}$ and $v' \in V_{\mathsf{source}}$, let $\mathsf{common\text{-}desc}_{(v,v')}$ be a boolean that denotes whether v and v' have a common descendant. That is, we define $\mathsf{common\text{-}desc}_{(v,v')} := (\mathsf{Desc}(v) \cap \mathsf{Desc}(v') \neq \emptyset)$

(f) If there is a $v \in V_{\mathsf{finalize}}$ such that for all other $v' \in V_{\mathsf{source}}$,

- $\mathsf{common\text{-}desc}_{(v,v')} = \mathsf{true}$
- Either $\mathsf{nDesc}(v) > \mathsf{nDesc}(v')$ holds or $(\mathsf{nDesc}(v) = \mathsf{nDesc}(v')) \wedge (\mathsf{TXS}(v), \mathsf{TXS}(v')) \in \mathsf{Ord}$.

then, deliver transactions in v by appending $\mathsf{TXS}(v)$ to LOG_i. Remove v from G_i^* and the corresponding vertices form G_i.

(g) Repeat steps 4b to 4f until there is no such v in step 4f.

(h) Output the current LOG_i to $\mathcal{Z}$.

• When i receives txs from $\Pi_{\text{fifocast}}[(\mathsf{sid}, j)]$, it appends txs to Log_i^j and adds j to the set U_i^{tx}.

• When i receives the output from $\Pi_{\text{sba}}[(\mathsf{sid}, \text{tx})]$, it stores it as L^{tx}.

Transaction Lifecycle. Suppose that a transaction tx is input to node i in round r_0. Since the external network is synchronous, by round $r_0 + \delta_{\mathsf{ext}}$, all nodes will have been input tx by $\mathcal{Z}$. Consequently, by round $r_1 = r_0 + \delta_{\mathsf{ext}} + T_{\text{confirm}}^{\text{fifocast}}$, node i will have received the gossip broadcasts from all other honest nodes. By round $r_2 = r_1 + T_{\text{confirm}}^{\text{sba}}$, node i will receive the output of the agreement stage for tx, and tx can be added to the graph G_i. Now by round $r_3 = r_2 + 2\delta_{\mathsf{ext}}$, any

other transaction that tx could be incomparable with will also get added to G_i. Waiting for this time ensures that tx does not get delivered before ensuring that all relevant transactions have been placed in the graph.

We now show the consistency (in Section 3.3.3), weak-liveness (in Section 3.3.4), and batch-order-fairness (in Section 3.3.5) properties for the $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ protocol. Proofs are deferred to Appendix B.2.

3.3.3 Consistency

To show consistency, we need to prove that two honest nodes i and j remove transactions from their graphs G_i^* and G_j^* in the same order. For this, we first present a helpful lemma (Lemma 3.8), showing that the graphs G_i^* and G_j^* for honest nodes i and j get built in the same way.

Lemma 3.8. *Suppose that when an honest node i delivers* tx, $v = \mathsf{SCC}_i(\mathsf{tx})$ *is the vertex that contains* tx *in* G_i^*. *That is,* tx *is delivered in the set of transactions* $\mathsf{TXS}_j(v)$. *Now, if another honest node j delivers* tx *and* $v' = \mathsf{SCC}_j(\mathsf{tx})$ *at that point, then* $\mathsf{TXS}_i(v) = \mathsf{TXS}_j(v')$, *or equivalently* $\mathsf{SCC}_i(\mathsf{tx}) = \mathsf{SCC}_j(\mathsf{tx})$ *when* tx *is output by each of the nodes. This means that we can drop the node subscripts.*

We can now state the consistency theorem.

Theorem 3.9 (Consistency of $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$)**.** *Consider any* $n, f, \gamma, \Delta_{\mathsf{ext}} = (\mathsf{full}, \delta_{\mathsf{ext}}), \Delta_{\mathsf{int}} = (\mathsf{full}, \delta_{\mathsf{int}})$ *with* $n > \frac{2f}{2\gamma-1}$. *Let* Π_{fifocast} *be a secure* FIFO-BC *protocol and* Π_{sba} *be a secure* Set-BA *protocol. Then,* $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ *satisfies consistency w.r.t. any* $(\mathcal{A}, \mathcal{Z})$ *that respects* $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$*-classical execution.*

Both proofs are deferred to Appendix B.2.1.

3.3.4 Liveness

As mentioned before, we show that $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ satisfies weak-liveness. To show weak-liveness for a transaction tx, first, in Lemma 3.10, we prove that if a transaction is input sufficiently after tx, it cannot be coalesced into the same strongly connected component as tx.

Lemma 3.10. *Consider a transaction* tx *and build the set* T *as per the weak-liveness definition. That is,* T *is built recursively as follows: (1) Add* tx *to* T*; (2) For* $\mathsf{tx}^* \in \mathsf{T}$*, add to* T*, all transactions that were received by at least one honest node before* tx^**. Now, suppose that there was another transaction* tx' *that is input to all nodes after all transactions in* T*. Then* $\mathsf{SCC}_i(\mathsf{tx}) \neq \mathsf{SCC}_i(\mathsf{tx}')$ *for any honest* i*.*

We now present the weak-liveness result for $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$.

Theorem 3.11 (Weak-Liveness of $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$)**.** *Consider any* n, f, γ, $\Delta_{\mathsf{ext}} = (\mathsf{full}, \delta_{\mathsf{ext}})$, $\Delta_{\mathsf{int}} = (\mathsf{full}, \delta_{\mathsf{int}})$ *with* $n > \frac{2f}{2\gamma-1}$*. Let* Π_{fifocast} *be a secure* FIFO-BC *protocol and* Π_{sba} *be a secure* Set-BA *protocol. Further, suppose that* Π_{fifocast} *satisfies* $(T^{\text{fifocast}}_{\text{warmup}}, T^{\text{fifocast}}_{\text{confirm}})$ *liveness, and* Π_{sba} *satisfies* $T^{\text{sba}}_{\text{confirm}}$ *liveness. Then,* $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ *satisfies* $(T^{\text{fifocast}}_{\text{warmup}}, T^{*}_{\text{confirm}})$*-weak-liveness where* $T^{*}_{\text{confirm}} = 5\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}} - 1$ *w.r.t. any* $(\mathcal{A}, \mathcal{Z})$ *that respects* $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$*-classical execution.*

Both proofs are deferred to Appendix B.2.2.

In Section 3.3.6, we show a simple modification to $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ protocol achieves (conventional) liveness, while continuing to achieve consistency and order-fairness, when the external network is restricted such that $\delta_{\mathsf{ext}} \leq 1$.

3.3.5 Batch-order-Fairness

Theorem 3.12 (Batch-order-Fairness of $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$)**.** *Consider any* $n, f, \gamma, \Delta_{\mathsf{ext}} = (\mathsf{full}, \delta_{\mathsf{ext}}), \Delta_{\mathsf{int}} = (\mathsf{full}, \delta_{\mathsf{int}})$ *with* $n > \frac{2f}{2\gamma-1}$. *Let* Π_{fifocast} *be a secure* FIFO-BC *protocol and* Π_{sba} *be a secure* Set-BA *protocol. Further, suppose that* Π_{fifocast} *satisfies* $(T^{\text{fifocast}}_{\text{warmup}}, T^{\text{fifocast}}_{\text{confirm}})$ *liveness, and* Π_{sba} *satisfies* $T^{\text{sba}}_{\text{confirm}}$ *liveness. Then,* $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ *satisfies* $(\gamma, T^{\text{fifocast}}_{\text{warmup}})$ *batch-order-fairness w.r.t. any* $(\mathcal{A}, \mathcal{Z})$ *that respects* $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$*-classical execution.*

Corollary 3.12.1. $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ *also satisfies* $(\gamma, T^{\text{fifocast}}_{\text{warmup}})$ *receive-order-fairness when* $\delta_{\mathsf{ext}} \leq 1$.

Both proofs are deferred to Appendix B.2.3.

3.3.6 Modified protocol for $\delta_{\mathsf{ext}} \leq 1$

In this section, we show a modification to the $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ protocol that achieves (conventional) liveness when $\delta_{\mathsf{ext}} \leq 1$.

Consider a protocol Π^* that modifies $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ as follows: For a transaction tx input in round r, in the finalization step, instead of waiting for a common descendant, deliver tx in round $r + 4\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}}$ based on which of its incomparable vertices have a larger number of descendants. Specifically, Π^* simply excludes the check for $\mathsf{common\text{-}desc}_{(v,v')} = \mathsf{true}$ in step 4f of the pseudocode for $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ (Section 3.3.2).

Theorem 3.13 shows that Π^* satisfies consistency, (conventional) liveness, and receive-order-fairness.

Theorem 3.13. *Consider any* n, f, γ, $\Delta_{\mathsf{ext}} = (\mathsf{full}, \delta_{\mathsf{ext}})$, $\Delta_{\mathsf{int}} = (\mathsf{full}, \delta_{\mathsf{int}})$ *with* $n > \frac{2f}{2\gamma-1}$. *Let* Π_{fifocast} *be a secure* FIFO-BC *protocol and* Π_{sba} *be a secure* Set-BA *protocol. Further, suppose that* Π_{fifocast} *satisfies* $(T^{\text{fifocast}}_{\text{warmup}}, T^{\text{fifocast}}_{\text{confirm}})$ *liveness, and* Π_{sba} *satisfies* $T^{\text{sba}}_{\text{confirm}}$ *liveness. Then, the protocol* Π^* *(modified from* $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$*) satisfies consistency,* $(T^{\text{fifocast}}_{\text{warmup}}, T^*_{\text{confirm}})$*-liveness where* $T^*_{\text{confirm}} = 5\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}}$*, and* $(\gamma, T^{\text{fifocast}}_{\text{warmup}})$*-receive-order-fairness w.r.t. any* $(\mathcal{A}, \mathcal{Z})$ *that respects* $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$*-classical execution.*

Proof. The proof is deferred to Appendix B.2.4. □

3.4 The Asynchronous Aequitas protocol

We describe the leaderless asynchronous protocol $\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$ in this section. The general technique is the same as the one for its synchronous equivalent which we discussed in Section 3.3.1. We note the major modifications here:

- First, we note that we can no longer wait for a specific number of rounds, since we are not making any synchrony assumptions. Rather, to start the agreement phase, a node i waits to receive a transaction $n - f$ times from other nodes. This means that after the agreement phase for tx returns the set L^{tx}, only $n - 2f$ indices are guaranteed to be honest (instead of $n - f$ honest in the synchronous protocol).
- Now, to realize $(\gamma, T_{\text{warmup}})$ batch-order-fairness, we need $\gamma n - 2f > \frac{n}{2}$ to hold. Equivalently, we need $n > \frac{4f}{2\gamma-1}$. This means that even for $\gamma = 1$, our protocol requires $n > 4f$. For other values of γ, the fraction of nodes allowed to be corrupted will be smaller.

- When i receives the output L^{tx} from the agreement phase, it makes sure that any transaction that appears in at least $f + 1$ logs is added to the graph G_i. This ensures that before delivering tx, any other vertex that it is "incomparable" with exists in the graph. At the same time, it also ensures that maliciously placed transactions are not added to the graph.
- Suppose that v and v' are incomparable. The synchronous protocol delivered the vertex with the higher number of descendants when a common descendant v_{common} was added to the graph. This is not enough to ensure consistent ordering across nodes in the asynchronous setting. Before delivering v or v', a node also needs to wait for vertices incomparable with v_{common} to be received. Note that this does not happen within a fixed time as it did in the synchronous case.

The rest of the protocol is more or less identical to the synchronous protocol $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$. For completeness, we write down the entire protocol in Section 3.4.1. In the subsequent sections, we show that $\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$ satisfies consistency, weak-liveness, and order-fairness.

3.4.1 Protocol Pseudocode

We describe $\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$ for $\gamma = 1$ for an honest node i below:

- **(Gossip)** When i receives a set of transactions txs from $\mathcal{Z}$, it does the following:
  1. Fork an instance of $\Pi_{\text{fifocast}}[(\mathsf{sid}, i)]$ with i as the sender, if it does not already exist.

2. Send txs as input to $\Pi_{\text{fifocast}}[(\mathsf{sid}, i)]$.

- When i receives txs from $\Pi_{\text{fifocast}}[(\mathsf{sid}, j)]$, it does the following:

  1. Append txs to Log_i^j and add j to the set U_i^{tx}.
  2. **if** $|U_i^{\text{tx}}| \geq n - f$, **then** fork an instance of $\Pi_{\mathsf{Set\text{-}BA}}[(\mathsf{sid}, \text{tx})]$ and provide it the input U_i^{tx}

- When i receives L^{tx} from $\Pi_{\text{sba}}[(\mathsf{sid}, \text{tx})]$, it does the following:

  1. Record the output L^{tx}
  2. Add a vertex denoted by tx to G_i if it does not already exist
  3. For any tx′ seen in at least $f+1$ Log_i^j, add tx′ to G_i if it does not already exist
  4. For any tx′ in G_i, if $L^{\text{tx}'}$ exists, calculate $l_{(\text{tx},\text{tx}')}$ as per Section 3.2.1. If $l_{(\text{tx},\text{tx}')} \leq f$, add the edge (tx', tx) to G_i
  5. Run the Finalization step

- **(Finalization)**

  1. Compute the *condensation* graph G_i^* of G_i by collapsing each strongly connected component into a single vertex.
  2. Let V_{source} be the set of vertices in G_i^* where $v \in V_{\mathsf{source}}$ if it satisfies:
     - All transactions in $\mathsf{TXS}(v)$ have been received.
     - v is a source vertex in G_i^*. That is, v has no incoming edges.
  3. For $v \in V_{\mathsf{source}}$, let $\mathsf{Desc}(v)$ denote the descendants of v in G_i^*. Let $\mathsf{nDesc}(v) = |\mathsf{Desc}(v)|$ i.e. the number of descendants.

4. For $v, v' \in V_{\mathsf{source}}$, let $\mathsf{common\text{-}desc}_{(v,v')}$ be a boolean that denotes whether v and v' have a common descendant. That is, we define $\mathsf{common\text{-}desc}_{(v,v')} := (\mathsf{Desc}(v) \cap \mathsf{Desc}(v') \neq \emptyset)$

5. If there is a $v \in V_{\mathsf{source}}$ such that for all $v' \in V_{\mathsf{source}}$,

   - $\mathsf{common\text{-}desc}_{(v,v')} = \mathsf{true}$
   - Suppose that v_{common} is the common descendant. Then, check that the transactions in any vertex incomparable with v_{common} has been received.
   - Either $\mathsf{nDesc}(v) > \mathsf{nDesc}(v')$ holds or $(\mathsf{nDesc}(v) = \mathsf{nDesc}(v')) \wedge (\mathsf{TXS}(v), \mathsf{TXS}(v')) \in \mathsf{Ord}$.

   then, deliver transactions in v by appending $\mathsf{TXS}(v)$ to LOG_i. Remove v from G_i^* and the corresponding vertices form G_i.

6. Repeat steps 2 to 5 until there is no such v in step 5.

7. Output the current LOG to $\mathcal{Z}$.

3.4.2 Asynchronous Aequitas Properties

We now state the properties of the asynchronous Aequitas protocol. The proofs are deferred to Appendix B.3.

Theorem 3.14 ($\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$ properties)**.** *Consider any* $n, f < \frac{n}{4}, \Delta_{\mathsf{ext}} = (\mathsf{none}, \infty), \Delta_{\mathsf{int}} = (\mathsf{none}, \infty), \gamma = 1$. *Let* Π_{fifocast} *be a secure* FIFO-BC *protocol and* Π_{sba} *be a secure* Set-BA *protocol. Then,* $\Pi^{\mathsf{async,nolead}}_{\mathsf{Aequitas}}$ *satisfies consistency, eventual-weak-liveness, and* $(\gamma, T_{\mathrm{warmup}})$ *batch-order-fairness w.r.t. any* $(\mathcal{A}, \mathcal{Z})$ *that respects* $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$*-classical execution.*

3.5 Leader-Based Aequitas Protocols

We use this section to describe a sketch of the leader-based Aequitas constructions, $\Pi^{\mathsf{sync,lead}}_{\mathsf{Aequitas}}$ and $\Pi^{\mathsf{async,lead}}_{\mathsf{Aequitas}}$. For this, we will pair an existing leader-based consensus protocol Π_{leader} with the three Aequitas stages described in Section 3.2.

The Aequitas stages. Each node follows the three stages of the Aequitas protocol. An honest node i broadcasts or "gossips" transactions as it receives them from the environment. Next, all nodes agree on which of these broadcasts to use to determine the ordering for a particular transaction. Finally, i builds the "waiting" graph G_i.

Leader proposal. The actual method of selecting the leader is orthogonal to our construction. Leaders may be cycled periodically or only when there is a detected failure. We only assume that the current leader node is known to all nodes so that proposals from non-leader nodes can be immediately rejected. The current leader node proposes a set of blocks to add to the log. Suppose that we represent the proposal by $S_1, \ldots, S_p$ where each of the S_x are sets of transactions. Before accepting the proposal, an honest node does the following:

1. Use the protocol Π_{leader} to reach agreement on the block proposal (to ensure that the leader does not equivocate). During the voting for Π_{leader}, ensure that at least $f + 1$ nodes received the transactions in the proposal from $\mathcal{Z}$.
2. Ensure that the proposed transactions are valid.
3. For each set S_x from S_1 to S_p,

- Wait for all tx $\in S_x$ to be received and added to G_i. If all tx $\in S_x$ do not belong to the same strongly connected component in G_i, then reject the proposal.
- If $\mathsf{SCC}(\text{tx})$ has an incoming edge in G_i^* that has not been delivered from S_1 to S_{x-1}, then reject the proposal.

4. Accept the proposal and append $S_1, \ldots, S_p$ to LOG_i.
5. Remove the delivered transactions from G_i (and G_i^*).
6. Output LOG_i to $\mathcal{Z}$.

CHAPTER 4

A NEW SOCIAL CHOICE THEORY PERSPECTIVE ON ORDER-FAIRNESS

This chapter describes a new social choice and voting theory perspective on order-fairness, which arises as a result of distilling the key structure of the Aequitas protocol.

Recall that Aequitas functions in three stages—gossip, agreement, and finalization. The first two stages can broadly be thought of as the "consensus" part of the design. On the other hand, the finalization stage related more to the "ordering" part—it is where the aggregated transaction ordering gets decided.

Essentially, this means that we can abstract out the broadcast and agreement primitives used in the first two stages. As a result, for the ordering component, we provide a self-contained formulation where nodes start with the transaction input orderings of all nodes (of which f orderings may be adversarial chosen). This results in an interesting new connection of the finalization stage with social choice theory.

The content in this chapter was initially published at APKC 2022 with the title "Order-Fair Consensus in the Permissionless Setting" [157].

4.1 The Finalization Problem

Before we formally define the finalization problem, we introduce some notation. For our finalization problem parameterized by $\Upsilon = (n, f, \gamma)$, we will consider as input, n ordered lists List_1 to List_n where an entry in a list is a set of messages. While the inputs to the finalization problem are independent of any underlying

consensus problem, in the permissioned setting, intuitively, these lists describe the order in which messages were input to a particular node. We use $\mathsf{List}_i[r]$ to denote the the set of messages received by node i in round r (if i is honest). Lists corresponding to adversarial nodes, can of course deviate arbitrarily from their true input ordering. For messages m and m', we use the notation $m \prec_i m'$ if m is ordered before m' in List_i. The goal now, is to output a list $\mathsf{OutputList}$ of transactions that are ordered in a "fair" manner. We say that $\Upsilon = (n, f, \gamma)$ is admissible if $\frac{1}{2} < \gamma \leq 1$ and $n > \frac{2f}{2\gamma - 1}$. We define a problem instance by $\Lambda = (\Upsilon, \mathcal{L})$ where $\Upsilon = (n, f, \gamma)$ and $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_n]$ indexed by 1 to n.

Definition 4.1 (Valid Message)**.** We say that a message m is valid w.r.t. $\Lambda = (\Upsilon, \mathcal{L})$ if it is present in at least $\Upsilon.n - \Upsilon.f$ lists in $\mathcal{L}$.

Definition 4.2 (γ-Global Preference)**.** Given $\Lambda = (\Upsilon = (n, f, \gamma), \mathcal{L})$, for messages m and m', we say that m is *globally preferred* over m', denoted by $m' \lhd^{\Lambda} m$, if there are at least $\gamma n - f$ input lists List_j in $\mathcal{L}$ s.t. $m \prec_j m'$. We will write $m' \lhd m$ when Λ is clear from context.

We define Condorcet cycles for non-transitivity in the global preference, which can arise from the Condorcet paradox as discussed in the previous chapter. Intuitively, this also makes receive-order-fairness impossible in the general case resulting in us using the batch relaxation.

Definition 4.3 (Condorcet Cycle)**.** Given $\Lambda = (\Upsilon, \mathcal{L})$, we say that $(m_1, m_2, \ldots, m_l)$ is an l-length Condorcet cycle in $\mathcal{L}$ if $m_1 \lhd^{\Lambda} m_l$ and $m_{i+1} \lhd^{\Lambda} m_i$ for all $i \in \{1, \ldots, l\}$.

We start by describing a simpler static version of the finalization problem.

Problem 4.4. (Strong (resp. Weak) Static Fair-Order Finalization) Consider an admissible $\Lambda = (\Upsilon, \mathcal{L})$ as input. Output a list OutputList containing all valid messages in $\mathcal{L}$ such that m is ordered before (resp. no later than) m' in OutputList if m is globally preferred over m' w.r.t. Λ, or equivalently, $m' \lhd^{\Lambda} m$.

4.2 Static-Finalization as Social Choice Aggregation

It is easy to draw parallels between the Static Fair-Order Finalization problem (Problem 4.4) and majority voting systems, in which voters rank a set of candidates. In such a voting system, a candidate A is preferred over a candidate B if a majority of voters prefer A over B. The goal of a voting protocol now is to find the winning candidate(s), or more generally rank all candidates by "aggregating" voter preferences. For our purpose, we are essentially replacing candidates with transactions and voters with consensus nodes. We note that Problem 4.4 also generalizes the condition when one transaction is preferred over another. Instead of a simple majority, we parameterize preference using a order-fairness parameter γ (see Definition 4.2). We also allow some of the "voting preferences" to be adversarial, which is not something usually considered in voting theory.

Condorcet cycles and Smith sets. One standard choice for selecting the winner(s) in a majority voting election is the Smith criterion [234]. The Smith set is the smallest possible set of candidates that are preferred over other candidates not in the set. When a Condorcet winner exists (i.e., there are no Condorcet cycles), it is always the sole member of the Smith set. By a Condorcet cycle, we mean a sequence of candidates (or messages), where the global preference rela-

tion is non-transitive. Such cycles can exist even when individual orderings are transitive due to the Condorcet paradox [88, 161].

Definition 4.5 ((Majority) Smith Set)**.** The Smith set is the smallest set $\mathcal{S}$ that satisfies the condition that for all $x \in \mathcal{S}$ and $y \notin \mathcal{S}$, x is preferred over y in a majority of orderings.

The Smith set can be computed using standard graph based algorithms. By representing global preferences as directed edges in a graph (e.g., $x \to y$ if x is globally preferred over y), computing the Smith set is now equivalent to finding the strongly connected component with no incoming edges from outside of the component. The Smith set is therefore, sometimes referred to as the "top cycle." It can be found by using variations of the Floyd-Warshall algorithm or Tarjan's algorithm (see [89] for common graph algorithms). Similarly, all candidates can be ranked by topologically sorting the strongly connected components.

We note that it is not particularly difficult to compute the fair ordering in the static finalization problem. Indeed, the same techniques for computing the Smith set can be used to find the fair ordering for transactions. We detail the algorithm in Figure 4.1. When no Condorcet cycles exist in $\mathcal{L}$, this protocol solves the strong static finalization problem; otherwise, it solves the weak variant.

4.3 A New Streaming Social Choice Variant

The primary difference that separates the Aequitas finalization protocol from prior work in voting theory is that the list of preferences is not fully determined at the start of the protocol. In the consensus setting, the protocol is ongoing

Static Finalization Algorithm **StaticFinalize**

Input: $\Lambda = (\Upsilon, \mathcal{L})$ where $\Upsilon = (n, f, \gamma)$ is admissible and $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_n]$.
Notation: Let $\mathcal{V}$ be the set of valid messages in the input lists.
Algorithm Description.

1. Construct an empty graph $G = (V, E)$, where V denotes the set of vertices, and E denotes the set of edges. For all $m \in \mathcal{V}$, add a vertex labeled m to G.
2. For all $m, m' \in \mathcal{V}$, if $m' \lhd^{\Lambda} m$, then add the edge (m, m') to G.
3. Let G_{con} be the condensation graph of G. Output the topological sorting of G_{con}.

Figure 4.1: Static Finalization Algorithm

as new transactions are continuously input to nodes[1]. Clearly nodes should not have to wait for all transactions to be input to decide on the fair ordering. Here, future transactions can cause changes in the condensation graph. Similarly, current transactions may not have been seen sufficiently many times so far. This means that current transactions might need to wait for future ones in order to be output in a fair order. The key technical challenge is to identify when transactions can be delivered without compromising on their fair ordering. We formalize this as the streaming finalization problem (Problem 4.6).

For lists List and List', we use $\mathsf{List} \preccurlyeq \mathsf{List}'$ to denote that List is a prefix of List'. For $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_k]$ and $\mathcal{L}' = [\mathsf{List}'_1, \ldots, \mathsf{List}'_k]$, we also use the notation $\mathcal{L} \preccurlyeq \mathcal{L}'$ to denote that $\mathsf{List}_j \preccurlyeq \mathsf{List}'_j$ for all j.

Problem 4.6 (Strong (resp. Weak) Streaming Finalization)**.** Consider an admissible $\Upsilon = (n, f, \gamma)$. At each timestep r, consider as input $\mathcal{L}^r = [\mathsf{List}^r_1, \ldots, \mathsf{List}^r_n]$ such that $\mathcal{L}^{r_1} \preccurlyeq \mathcal{L}^{r_2}$ whenever $r_1 \leq r_2$. At timestep r, output a list $\mathsf{OutputList}^r$ such that:

[1]Intuitively, this is analogous to a growing list of potential candidates in an election, which is not particularly relevant for voting theory.

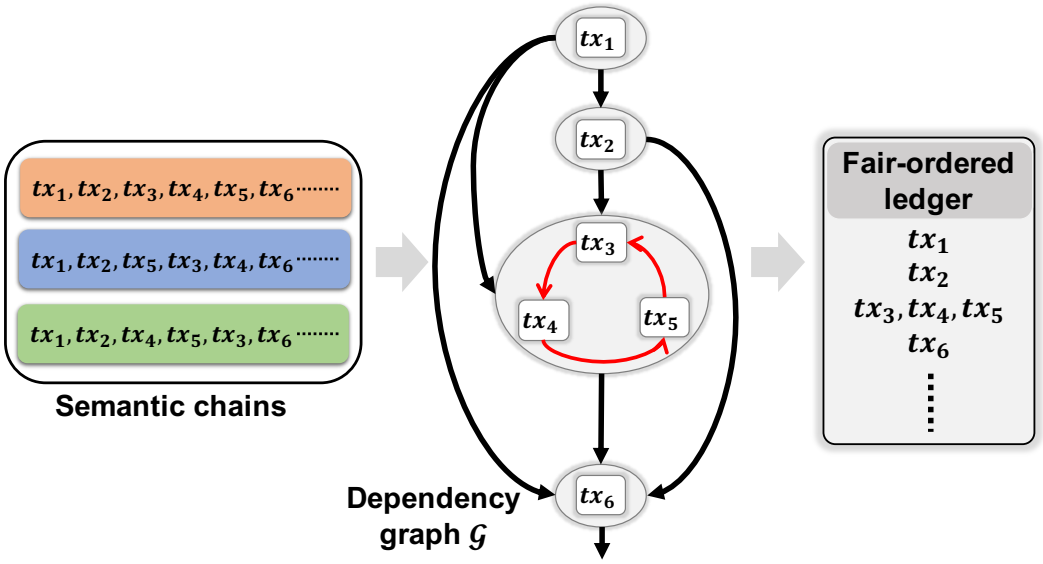

Figure 4.2: Simple Finalization Example

- (Prefix-Consistency) $\mathsf{OutputList}^{r_1} \preccurlyeq \mathsf{OutputList}^{r_2}$ whenever $r_1 \leq r_2$.
- (Strong (resp. Weak) Fairness of ordering) For all (m, m') such that $m' \triangleleft^{\Lambda} m$, where $\Lambda = (\Upsilon, \mathcal{L}^r)$, if $\mathsf{OutputList}^r$ contains m or m', then m is ordered before (resp. no later than) m'.

An algorithm $F(\cdot)$ solves the streaming finalization problem for an admissible Υ if for any valid input sequence $\mathcal{L}^1, \mathcal{L}^2, \ldots$, for every timestep r, $F(\mathcal{L}^r)$ satisfies prefix-consistency and fairness of ordering. Note that if $\mathcal{L} \preccurlyeq \mathcal{L}'$, then $m' \triangleleft^{(\Upsilon,\mathcal{L})} m$ implies $m' \triangleleft^{(\Upsilon,\mathcal{L}')} m$ since for any $\mathsf{List}_j \in \mathcal{L}$ where $m \prec m'$, the corresponding $\mathsf{List}'_j \in \mathcal{L}'$ will also have $m \prec m'$. In other words, the fairness of ordering condition will not contradict the prefix-consistency condition. Note that liveness is not included in Problem 4.6 (analogous to how delivering no transactions still satisfies "consistency" in the consensus abstraction). Liveness will be a separate protocol requirement.

4.3.1 Aequitas Solves Streaming Finalization

Given the n input orderings from the protocol nodes, the Aequitas finalization stage first builds a dependency graph of transactions where edges correspond to global preferences and computes its condensation graph by collapsing the strongly-connected-components.

For our purpose, we can abstract out the underlying protocol nodes and run the Aequitas finalization only on the lists. In general, given $\Upsilon = (n, f, \gamma)$ and $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_n]$, we will use $\mathsf{Graph}_\Upsilon(\mathcal{L})$ to denote the dependency graph and $\mathsf{Aequitas}_\Upsilon(\mathcal{L})$ to denote the resultant outputs of applying the finalization step of the Aequitas protocol on the lists in $\mathcal{L}$. We drop the subscript when Υ is clear from context. We provide a simple example what finalization looks like in Figure 4.2. Cycles represent non-transitive preferences that will be output in the same "batch" for batch-order-fairness. Recall that here, transactions within a batch can still be totally ordered for execution; the only difference is their ordering is no longer considered unfair.

Now, from the condensation graph, the transactions in a vertex v are delivered in one of two ways:

1. v is a source vertex and for all other vertices v', there is a path from v to v'
2. v is a source vertex and for any other vertex v' without a path from v, there is a common descendant of v and v' and either v has more descendants or v, v' have the same number of descendants but v is preferred in a previously agreed upon ordering relation (e.g., alphabetical ordering).

Intuitively, a transaction can be output if its strongly connected component

can no longer change due to other transactions and it has no incoming edge. Within Aequitas, whenever m is globally preferred to m', the finalization stage will order m no later than m' in the general case, and m strictly before m' when in a setting where there are no Condorcet cycles. In turn, this directly implies that the Aequitas finalization solves our (weak) streaming finalization abstraction where the input lists for honest nodes are the actual ordering of inputs from $\mathcal{Z}$.

Fact 4.7. For an admissible Υ, $\mathsf{Aequitas}_{\Upsilon}(\cdot)$ solves the weak streaming fair-order finalization problem.

Proof. First, from the self-consistency of the Aequitas protocol, we can infer that if $\mathcal{L} \preccurlyeq \mathcal{L}'$, then $\mathsf{Aequitas}_{\Upsilon}(\mathcal{L}) \preccurlyeq \mathsf{Aequitas}_{\Upsilon}(\mathcal{L}')$. Furthermore, $\mathsf{Aequitas}_{\Upsilon}(\cdot)$ satisfies the weak fairness of ordering property since the Aequitas protocol satisfies batch-order-fairness in the permissioned setting. □

Remark 1 (Minimally-relaxive batch-order-fairness)**.** We can also show that although $\mathsf{Aequitas}(\cdot)$ solves the weak fairness of ordering property, the version it satisfies is in some sense a minimally relaxed form of the strong fairness of ordering property. Specifically, when $m' \lhd^{\Lambda} m$, m will be ordered before m' *except* when there is a Condorcet cycle that contains both m and m'. In other words, m and m' will be output at the same index only when a paradoxical ordering prevents them from being ordered any other way. To see why this holds, recall that the Aequitas finalization protocol from KZGJ outputs m and m' together if an only if they are part of the same strongly-connected-component in the dependency graph. Consequently, in such a case, there is a path from m to m' and from m' to m which creates a cyclic ordering dependency.

CHAPTER 5

ORDER-FAIRNESS IN THE PERMISSIONLESS SETTING

In this chapter, we extend the notion of order-fairness to consensus protocols in ther permissionless setting—that is, settings where the set of protocol parties is not known and may dynamically change.

Throughout this chapter, we use KZGJ to denote the work on order-fair consensus by Kelkar, Zhang, Goldfeder, and Juels [161] (note that this forms Chapters 2 and 3 of this thesis).

The content in this chapter was initially published at APKC 2022 with the title "Order-Fair Consensus in the Permissionless Setting" [157].

5.1 Model and Preliminaries

We describe the general execution model in this section. Much of our formalism is adapted from an extensive line of research on permissionless consensus [129, 204, 203, 207, 208]. Further, it also extends the order-fairness formalism from Chapter 2 to the permissionless setting. Lifting the network and order-fairness formalism to the permissionless setting brings out several subtleties which we detail in Sections 5.1.1 and 5.1.4.

General Execution Model. As standard practice [129, 203], we model protocol nodes as ITMs [57], and use a special environment machine $\mathcal{Z}$ to direct execution. We assume a synchronous model where execution proceeds in *rounds*. At the start of a round, nodes receive transactions as input from $\mathcal{Z}$; at the end of

a round, nodes may deliver[1] outputs to $\mathcal{Z}$. For ease of modeling, for honest nodes, the final ordering will be taken as the result of a transformation function $\mathsf{linearize}(\cdot)$ (part of the protocol description) on the outputs to $\mathcal{Z}$ rather than the outputs themselves. This will extract the relevant information from the outputs into a total ordering. Similar formalism is present in e.g., [208]. New transactions are modeled as being input by $\mathcal{Z}$ to avoid explicitly having to model clients.

Corruptions. $\mathcal{Z}$ can spawn new nodes either as *corrupt* or *honest*. Honest nodes execute the protocol faithfully while corrupt nodes are controlled by an adversary $\mathcal{A}$, and can deviate arbitrarily. Existing nodes can also be corrupted, which gives $\mathcal{A}$ full access to them. $\mathcal{A}$ can kill corrupted nodes which removes them from execution.

We observe that if the adversary is able to corrupt and kill nodes at will, it can cause the network to "forget" the transaction input orderings received by earlier honest nodes. While this was not important for prior work, this is critically detrimental for any fair ordering protocol. Therefore, we will assume a *(mildly)-respawning-adversary* that limits the node churn. Informally, a (τ, R)-respawning[2] adversary can kill at most τ fraction nodes within any period of R rounds. This corresponds to the notion that $\mathcal{A}$ cannot quickly corrupt and kill many nodes. We will distinguish between the respawning parameter and the adversarial corruption parameter (see Definition 5.2).

[1] In this paper, we will use the terms *deliver* and *output* interchangeably.

[2] This is distinguished from an R-mild adversary in [206] where the adversary takes R rounds to *corrupt* a node. While the security properties of [206] crucially relied on the adversary not gaining control of nodes instantly, we can tolerate the adversary corrupting a node instantly, as long as it can't kill too many nodes quickly.

Notational conventions. κ denotes the security parameter. For a protocol Π, $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ represents the random variable for all possible execution traces of Π w.r.t. $(\mathcal{A}, \mathcal{Z})$. The possible executions arise from any randomness used by honest nodes, adversarially controlled nodes, and the environment. Any view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$ is a fully specified instance (including inputs, outputs, random coins etc.) of an execution trace.

5.1.1 Communication Networks

Standard formalism for consensus protocols models a single communication network amongst consensus nodes. The permissioned order-fairness formalism, however, models two separate networks: an external network through which clients send messages, and an internal network for node communication. The adversary is assumed to only be in charge of the internal network. This was important to meaningfully discuss fair ordering, as it allowed honest nodes to have an unmolested view of their local transaction input ordering. This modeling is less realistic in the permissionless setting however, where nodes usually communicate through a gossip-style network. Therefore, we resort to modeling a single network that in spirit, melds together both networks. This brings out several subtleties in the network modeling, which we remark on in Section 5.1.2. Here, we directly proceed with the formal network definition we use below.

Network Formalism. We assume a synchronous model of communication, such that messages sent are delivered within a (known) bounded number of rounds Δ (a function of κ). The adversary $\mathcal{A}$ can delay and reorder messages (subject to the bound Δ) but cannot drop messages. We define the synchrony

assumption below.

Definition 5.1 ((Δ-synchrony)**.** $(\mathcal{A}, \mathcal{Z})$ respects Δ-synchrony if for all view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following hold:

- (Input Synchrony) If an honest node receives tx as input from $\mathcal{Z}$ in round r, then in any round $r' \geq r + \Delta$, any honest node in the system at round r' will have already received tx as input from $\mathcal{Z}$. This includes any node that may have spawned in round r'.
- (Ledger Synchrony) If an honest node outputs tx to $\mathcal{Z}$ in round r (recall that this is not the same as being ordered in the final ledger), then in any round $r' \geq r + \Delta$, any honest node in the system in round r' will have already received tx as input from $\mathcal{Z}$. This includes any node that may have spawned in round r'.
- (Broadcast Synchrony) If an honest node sends a message in round r, then in any round $r' \geq r + \Delta$, any honest recipient node will have received the message by round $r + \Delta$. This includes any node that may have spawned in round r'.
- $\mathcal{Z}$ provides Δ to all nodes when they are spawned.

Remark 2 (Input Synchrony and DoS)**.** Similar to the external synchrony in the permissioned setting, input synchrony assumes that the network gossips all transactions. In a permissionless network, this could lead to a denial-of-service (DoS) attack wherein an adversary floods the network with bogus double-spends (e.g., transactions that spend the same tokens). The Bitcoin network mitigates this by not gossiping any double-spends. This can be done for our protocols as well. We emphasize that doing so does not actually invalidate the

order-fairness property of our protocols since we only require adversarial transactions mined in a PoW chain to be gossiped.

We also note further that the proposal in [155] to prevent fast double-spends (although it cannot guarantee that later double-spends are not sequenced earlier due to adversarial miners) involves gossiping all double-spends through the entire network, for which the authors argue that the performance penalty incurred by the network would not be significant.

5.1.2 Remark on the Network Model

As mentioned before, allowing the adversary to only control the internal network in the permissioned setting was important to ensure that each consensus node had an unmolested view of its transaction input ordering. For example, the first node that receives a client transaction should not be able to adversarially insert its own transaction and quickly gossip it to other nodes in an attempt to get it sequenced first. This is a form of "front-running" as described in [92]. The assumption that the adversary does not control the external network takes care of this problem. However, this puts the onus on the client to send its transaction to all nodes instead of just a few. While this is a perfectly reasonable assumption in a permissioned network where the nodes are known, it is difficult to justify it in a permissionless network, where nodes change over time and transactions usually propagate through a gossip-style network. Here, arguably there is no way to get around nodes inserting their own transactions during the gossip. Nevertheless, we emphasize that this still retains the notion that a transaction that is seen by a majority of the network first, should be sequenced first.

We further note that privacy techniques (such as encryption) can be used as an orthogonal and complementary technique to hide transaction data before the ordering is finalized.

Even within the permissioned formalism, we could very well have modeled a single network, but the additional modeling of the external network allowed an honest node's input ordering to be completely independent of the adversary. In practice, in a permissionless network, clients should send their transactions to as many nodes as they can so that most nodes receive them in an order not influenced by the adversary.

Another upshot of modeling two separate networks in KZGJ was that a transaction claimed by an adversarial node would not be considered by an honest node unless the transaction was received from the environment i.e. an honest client. This works well since now, an adversary cannot influence the global state by creating bogus transactions. Unfortunately, in a permissionless Proof-of-Work protocol, transactions can also be gossiped by mining a block that contains it and propagating the block through the network. It is difficult to determine the final ordering simply based on the local input orderings of all the nodes from clients (like was done in KZGJ) since participating nodes may constantly change in a permissionless system. On the other hand, if transaction orderings are based on the ordering in the ledger, it could potentially enable an adversarial transaction (that was not input to any honest node), to be sequenced earlier simply by being placed earlier in the ledger. To get around this, informally, we force the environment to deliver transactions included in the ledger to all nodes *as input* which allows us to talk meaningfully about transaction ordering. This is captured in the ledger synchrony part of the definition

All of this ends up blurring the distinction between the internal and external network from KZGJ, leading us to model a single communication network.

5.1.3 Permissionless Formalism

We recall the definition of the permissionless setting next, taken from [208], and add to it our respawning parameters.

Definition 5.2 ($(n, \rho, \Delta, \tau, R)$-permissionless environments)**.** We say that $(\mathcal{A}, \mathcal{Z})$ respects $(n, \rho, \Delta, \tau, R)$-permissionless execution w.r.t. a protocol Π if for every $\kappa \in \mathbb{N}$, $(\mathcal{A}, \mathcal{Z})$ respects Δ-synchrony, and for all view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following holds: (1) In each round, there are exactly[3] n nodes online in the system, of which at most ρn are corrupt; (2) Within any period of R rounds, at most τn notes are killed. We call such an adversary (τ, R)-respawning; (3) $\mathcal{Z}$ provides all nodes the parameters n, ρ, Δ, τ, R upon spawning. Note that n, Δ, R are functions of κ.

Blockchain formalism. An abstract blockchain [203] chain is an ordered sequence of blocks output by an honest node. Let $\mathsf{chain}[i]$ denote the vector of transactions contained in chain at index i. For a given chain and non-negative integers b and b', we use $\mathsf{chain}[b]$ to denote the b^{th} block of chain (indexed from 0), $\mathsf{chain}[:b]$ to denote the first b blocks, and $\mathsf{chain}[b:b']$ to denote the sequence of blocks from $\mathsf{chain}[b]$ to $\mathsf{chain}[b']$, both inclusive. We also use $\mathsf{chain}[-b]$ ($b \neq 0$) to denote the b^{th} block from the rear and $\mathsf{chain}[:-b]$ to denote chain except for the last b blocks.

[3] While we model a fixed n in each round similar to most blockchain papers, we note that our model can easily be extended to handle a varying n, using complementary techniques [128, 67] that are well understood in literature.

We assume familiarity with the (by now) standard requirements for blockchain protocols, and refer the reader to [129, 203, 208] for more details. Informally, for honest chains, we require (1) T-common-prefix which allows confirmation of all except the last T blocks; (2) (T, g_0, g_1)-chain-growth which signals a growth of at least T in T/g_0 rounds and at most T blocks in T/g_1 rounds; (3) (T, μ)-chain-quality, which means that within any T consecutive blocks, at least μT are mined by honest nodes. Our proofs will require these properties from previous well-studied constructions only in a black-box way. For completeness, we also give the details in Appendix C.1.

Abstract blockchain protocols directly satisfy SMR consistency and livenes (see [203, 208]). For our purpose, we define $(T_{\text{warmup}}, T_{\text{confirm}})$-liveness as follows: If an honest node is input m by $\mathcal{Z}$ in round $r \geq T_{\text{warmup}}$, then for any $r' \geq r + T_{\text{confirm}}$ and any honest node in round r' that outputs $\mathcal{X}$ to $\mathcal{Z}$, m is in $\mathsf{linearize}(\mathcal{X})$. When there is no warmup time, we simply write T_{confirm}-liveness.

Random Oracle. We assume that all nodes have access to a random function $H : \{0,1\}^* \to \{0,1\}^\kappa$. This is provided by two oracles: $\mathsf{H}(x)$ which outputs $H(x)$ and $\mathsf{H.ver}(x, y)$ which outputs 1 if $H(x) = y$ and 0 otherwise. In any round, all parties can make q queries to H and any number of queries to $\mathsf{H.ver}$. An adversary $\mathcal{A}$ controlling p parties is allowed to make pq sequential oracle queries. Note that $\mathcal{Z}$ cannot access H, but can instruct $\mathcal{A}$ to make queries. This abstraction follows [129, 203], and captures the intuition that finding a proof-of-work solution is difficult, while checking the validity of a solution is inexpensive.

Blocks. A block $\mathtt{B}$ is a tuple $(h_{-1}, h', \mathtt{m}, \eta, h)$, where h_{-1} denotes a pointer to a parent block, h' denotes a pointer to a reference block (an arbitrary previous

block), $\mathtt{m}$ denotes the block record (eg., a vector of transactions), η denotes the PoW nonce, and h denotes the hash of the current block. The reference block is an important modeling choice since it allows $\mathtt{B}$ to *logically* follow a different block than its preceding parent block. Notably, this allows the validity condition for a block to also be defined in terms of the reference block, and possibly separately from the parent block. We note that $h_{-1} = h'$ in many cases (eg., the Nakamoto protocol).

Protocols are parameterized by a hardness function $p(\cdot)$ that defines $D_p = p(\kappa) \cdot 2^{\kappa}$ such that $\Pr_{\eta}[H(h_{-1}, h', \mathtt{m}, \eta) < D_p] = p(\kappa)$ for all $(h_{-1}, h', \mathtt{m})$. Now, we define a valid block $\mathtt{B}$ as a tuple $(h_{-1}, h', \mathtt{m}, \eta, h)$ where $H(h_{-1}, h', \mathtt{m}, \eta) = h$ and $h < D_p$.

We use $\Pi_{\mathsf{nak}}(p)$ to denote Nakamoto's protocol and $\Gamma^{p}_{\mathsf{nak}}(n, \beta, \Delta) = 1$ to denote a Nakamoto-compliant [203] execution. For completeness, we also provide the details in Appendix C.1.

5.1.4 Order-Fairness

While KZGJ studied fair transaction ordering in the permissioned setting, since almost all of the aforementioned real-world order-manipulation attacks are on permissionless systems, it is important to extend the order-fairness definition to the permissionless setting. We will do exactly that in this section. We refer to Sections 2.4 and 2.5 for a comparison of our fairness definition with other related ordering notions.

Choosing an order-fairness definition in the permissionless setting. Consider two transactions tx and tx′. First, we note that in the permissionless setting, nodes that are spawned far later (eg., after tx and tx′ are output), will receive tx and tx′ in the same round (see the input synchrony definition). Consequently, we cannot naïvely use the same order-fairness definition from the permissioned setting, since the antecedent will never be satisfied making the definition vacuous. Instead, we only consider the first time that the n nodes in the system receive either transaction, corresponding to the initial propagation of the transactions.

Let $\mathcal{S}$ be the set of these nodes. We say that a node has received tx before tx′ if it has either (1) received tx but not tx′ or (2) received both transactions and received tx in an earlier round than tx′. Now, if a large fraction (parameterized by γ) of nodes in $\mathcal{S}$ have received tx before tx′, then intuitively, tx should be preferred over tx′. For receive-order-fairness (resp. batch-order-fairness), this means that the final log will contain tx before (resp. no later than) tx′. Henceforth, for brevity, when we say that the final log contains tx before tx′, we also include the scenario where the log only contains tx. Permissionless order-fairness is formalized in Definition 5.3.

Definition 5.3 (Order-Fairness)**.** For a given view in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, we define γ-receive-order-fairness (resp. γ-batch-order-fairness) as follows: For two transactions tx and tx′, let r be the first round such that at least n nodes alive in round r, have received either tx or tx′. Now, if γ fraction of those nodes have received tx before tx′ as input, then for any round $t \geq r$, and any node i that is honest in round t and outputs $\mathcal{X}$, the final linearly ordered log, $\mathsf{linearize}(\mathcal{X})$, either (1) contains neither transaction; (2) contains only tx; or (3) contains tx before (resp. no later than) tx′.

We say that a protocol satisfies order-fairness if it satisfies the above, except with negligible probability over a random view.

Similar to KZGJ, the batch relaxation is used since KZGJ found that receive-order-fairness is impossible to achieve except in very specific settings. Note that for batch-order-fairness, transactions within a batch can still be totally ordered for execution; the only difference is that ordering within a batch is no longer considered unfair. The relaxation also is somewhat minimal in the sense that it is used on when the stronger notion is impossible.

5.2 Protocol Overview

To better motivate our design, we first take a step back to look at transaction ordering in standard Nakamoto consensus. Here, when the adversary mines a block, it has full control over the inclusion, exclusion, and ordering of transactions in the mempool. This essentially results in an *ephemeral* ordering centralization. To achieve fairness, we need to somehow decentralize the ordering process; this forms the basis for our design.

The key idea we use to get around this unilateral proposition of ordering of transactions is the following: ordering for each transaction (w.r.t. other transactions) is proposed across multiple, say d blocks. The proposed orderings within these blocks are then aggregated in a *fair* way to obtain the final transaction ordering. The miners of each of these d blocks report their own preferences on how to order a set of transactions — and honest nodes will report their true received order of transactions from the peer-to-peer network, while adversarial nodes can arrange transactions in arbitrary order in the block it mines so as to

influence the final ordering obtained from consensus. Owing to network delay or adversarial behavior in the network, different nodes might receive the same set of transactions from the p2p network in different orders and at different times. These differences will get reflected in the order in which the transactions are arranged in each of the d orderings that were used for reaching consensus on their final ordering. We call this design structure Hammurabi.

Our constructions follow this basic outline. In essence, the system as a whole, through e.g., PoW mining, will build d independent transaction orderings, which will be appended to as new transactions arrive. We call these orderings "semantic chains" since transactions in the same list are logically or semantically connected even though they may directly follow a different mined block. All transactions will be mined d times, once in each semantic chain. The final ordering will be extracted from the d semantic chains. Notably, d is a function only of the security parameter κ and not of the number of nodes. The upshot of such an approach is that in our protocols, the adversary will not be able to manipulate the ordering within a majority of semantic chains, resulting in a fair overall ordering. Figure 5.1 highlights this key idea as compared to existing protocols.

In the subsections that follow, we describe two protocols to form these semantic chains. Looking ahead, we will use the streaming finalization algorithm $\mathsf{Aequitas}(\cdot)$ to extract a fair final ordering from the semantic chains. For this, we prove a novel player replaceability lemma which will allows us to implement $\mathsf{Aequitas}(\cdot)$ in the permissionless setting.

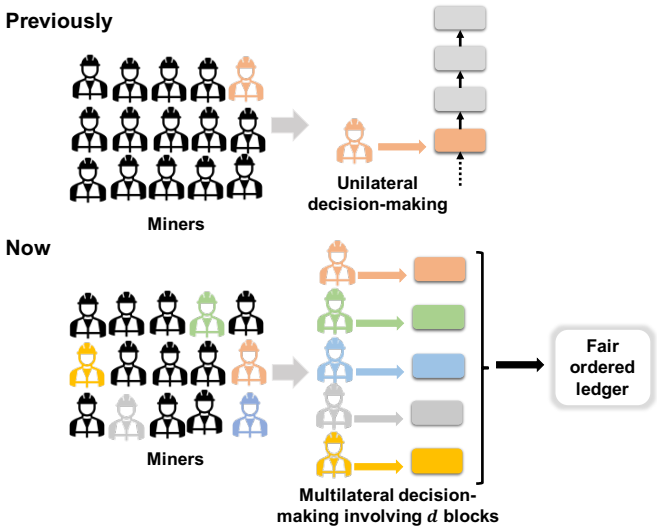

Figure 5.1: Permissionless Order-Fair Protocol Design

5.2.1 Player Replaceability

We now show how our streaming finalization abstraction assists in building order-fair protocols in the permissionless setting.

Recall that for the permissioned setting, the input lists used for Aequitas($\mathcal{L}$) were the actual transaction input orderings (for honest nodes). The key idea we use to move to the permissionless setting is to enable nodes to "continue" the orderings of other nodes in the system. In particular, we will fix a number of input lists, say d, at the start of the protocol. All nodes now extend the existing transaction orderings in these d lists. This can be done through any permissionless consensus technique (e.g., PoW, PoS etc). The d lists can be thought of as the input orderings of d "virtual" nodes in a permissioned network. The Aequitas finalization algorithm will now be run on the d lists to determine the final output ordering. To allow for a seamless reduction of the permissionless problem to a permissioned one, we use a new *player replaceability* lemma. Intuitively, this lemma will directly let us conclude order-fairness for our permissionless

protocols, by proving order-fairness for the permissioned transformation with d virtual parties.

Transformation property. The ability to use the "virtual" party transformation can be formalized as a completeness property of the dependency graph. Informally, there should be an edge from tx to tx′ in the dependency graph of any honest node whenever $(\mathsf{tx}, \mathsf{tx}')$ satisfies the antecedent of the order-fairness definition. Note that we consider the graph built from the full output $\mathcal{L}$ of the node and not the graph that it retains after delivering other transactions. This allows us to discuss an edge between transactions even though one of them might have been delivered in the final ledger.

Property 5.4 (Completeness of Dependencies)**.** Consider $(n, \beta, \Delta, \tau, R)$ and an order-fairness parameter γ. We say that Π is γ-*dependency-complete* if for any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$-permissionless execution, the following holds: If $(\mathsf{tx}, \mathsf{tx}')$ satisfies the antecedent of the γ-batch-order-fairness definition, then for honest node N in any round r, if N outputs $\mathcal{L}$ where tx′ is in all lists in $\mathcal{L}$, then $\mathsf{tx}' \lhd^{\Lambda} \mathsf{tx}$ holds where $\Lambda = (\Upsilon = (|\mathcal{L}|, \lceil \frac{|\mathcal{L}|}{2} \rceil - 1, 1), \mathcal{L})$. In other words, $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$ contains the edge $(\mathsf{tx}, \mathsf{tx}')$.

We can now describe the player replaceability lemma. We provide a simpler version that suffices for our purpose here, but prove a more general result in Appendix C.2.

Lemma 5.5 (Player Replaceability)**.** *Consider an $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \cdot, \cdot)$-permissionless execution and an order-fairness parameter γ. Suppose that a protocol Π is γ-dependency-complete and satisfies the following consistency-like property (from Problem 4.6):*

- *For any node N that is honest in a round r, N outputs $\mathcal{L}_N^r = [\mathsf{List}_1^r, \ldots, \mathsf{List}_d^r]$ to $\mathcal{Z}$.*
- *If N is honest in any rounds $r-1$ and r, then $\mathcal{L}_N^{r-1} \preccurlyeq \mathcal{L}_N^r$.*

Suppose that an algorithm $F(\cdot)$ satisfies the strong (resp. weak) streaming finalization problem for Υ and that the linearize *function of Π is defined to be $F(\mathcal{L}_N^r)$ for node N and round r. Then, Π satisfies γ-receive-order-fairness (resp. γ-batch-order-fairness) w.r.t. $(\mathcal{A}, \mathcal{Z})$.*

Intuitively, as long as a protocol Π that satisfies the specified properties (i.e., it provides a transformation to d virtual parties), the upshot of this lemma is that now, we can run $F(\cdot) = \mathsf{Aequitas}(\cdot)$ on the d semantic chains to determine the final fair output ordering. In Section 5.4, we show why our protocols confer to the prerequisite of Lemma 5.5 thereby achieving batch-order-fairness.

Remark 3 (Receive-Order-Fairness)**.** KZGJ provided a synchronous permissioned protocol that achieves receive-order-fairness when the synchrony parameter in the external network (between clients and nodes) was $\Delta_{\mathsf{ext}} = 1$. This is in part because an adversary can no longer claim to have received transactions too far apart from honest users (otherwise they would be easily detected), thereby resulting in no Condorcet cycles. For our purpose, since the adversary's orderings become intermingled with the honest ones through the PoW mining, attempting to realize receive-order-fairness would require rewinding PoW chains which might lead to other complications.

Although the impossibility result for receive-order-fairness from KZGJ when $\Delta_{\mathsf{ext}} > 1$ (and the adversary is allowed to corrupt at least one party) carries over

to the permissionless setting, we leave it as an open problem to resolve whether receive-order-fairness is achievable in the permissionless setting when $\Delta = 1$.

5.3 Permissionless Protocols

5.3.1 Modulo-T Longest Chain Protocol

We start with a basic order-fair protocol $\Pi^{\kappa,T}_{\text{mod}}$ which uses a single PoW-style longest chain, where κ is the security parameter, and T is the modular parameter. The protocol uses $d = T$ semantic chains but we use T here since it also corresponds to the consistency parameter of the underlying chain.

Protocol description. In $\Pi^{\kappa,T}_{\text{mod}}$, nodes mine a single PoW chain. For a given chain, $\mathsf{chain}[b]$ is considered to logically follow $\mathsf{chain}[b-T]$, despite its PoW hash following $\mathsf{chain}[b-1]$. In other words, the PoW chain will be split into T same semantic or "modular" chains, with blocks in the same index modulo T (e.g., indices 1, $T+1$, $2T+1$ and so on) describing a single transaction ordering. To obtain the fair ordering, $\mathsf{Aequitas}(\cdot)$ will be run on the semantic chains in the confirmed part of the PoW chain.

For a given chain, let $\mathcal{T}_b$ be the set of transactions that were first seen in block $\mathsf{chain}[b]$. Now, if $\mathsf{chain}[b]$ was honestly mined, then the first honestly mined block after $\mathsf{chain}[b]$ in each modular chain should contain these transactions. In other words, all honest nodes can reject blocks in the range $\mathsf{chain}[b+1 : b+T-1]$ that do not contain all transactions in $\mathcal{T}_b$. Consequently, in a valid chain, the record $\mathtt{m}$ in any block $\mathsf{chain}[b]$ should be such that $\bigcup_{i=b-T+1}^{b-1} \mathcal{T}_i \subseteq \mathtt{m}$. Figure 5.3 contains

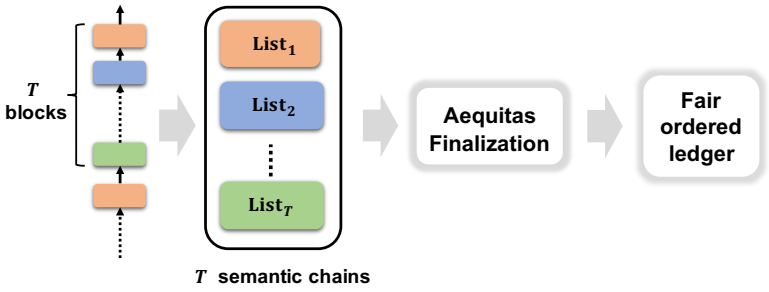

Figure 5.2: Pictorial representation of $\Pi_{\text{mod}}^{\kappa,T}$

a detailed description of Π_{mod} and Figure 5.2 contains a simplified pictorial representation.

Valid chain. A chain is valid iff (1) $\mathsf{chain}[0] = \mathit{genesis} = (0, 0, \perp, 0, H(0, 0, \perp, 0))$ is the genesis block; (2) $\mathsf{chain}[b].h_{-1} = \mathsf{chain}[b-1].h$ for all $b \in \{1, \cdots, |\mathsf{chain}|\}$; (3) $\mathsf{chain}[b].h' = \mathit{genesis}.h$ for $b \in \{1, \ldots, T\}$ and $\mathsf{chain}[b-T].h$ for all $b \in \{T+1, \cdots, |\mathsf{chain}|\}$; and (4) $\bigcup_{i=b-T+1}^{b-1} \mathcal{T}_i \subseteq \mathsf{chain}[b].\mathtt{m}$ for $b \in \{1, \cdots, |\mathsf{chain}|\}$.

Modular chains. For a given valid chain, we say that two blocks $\mathsf{chain}[b]$ and $\mathsf{chain}[b']$ where b, b' are positive integers are in the same modulo-T chain if $b \equiv b' \mod T$. The genesis block, $\mathsf{chain}[0]$ is assumed to be included in all modular chains. We use $\mathsf{modularize}(\mathsf{chain}, T)$ to denote the operation of splitting chain into modulo-T chains. Specifically, the output of $\mathsf{modularize}(\mathsf{chain}, T)$ is $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_T]$ such that List_j concatenates the transaction orderings in all $\mathsf{chain}[b]$ where $b \equiv j \mod T$. We will use $\mathtt{ref} = h'$ to point to the previous block in the same modular chain.

Dynamic corruption. Recall that we restrict $\mathcal{A}$ to be (τ, R)-respawning, i.e., at most τn nodes can be killed within any R round period. Now, for $\Pi_{\text{mod}}^{\kappa,T}$, we will

$\Pi^{\kappa,T}_{\text{mod}}(p)$

Initialize: chain = *genesis*; TxPool $= \emptyset$

Upon receiving a valid chain$'$ such that $|\text{chain}'| > |\text{chain}|$:

- Set chain := chain$'$ and output any newly seen transactions to $\mathcal{Z}$.

In round r, upon receiving txs from $\mathcal{Z}$:

- For each tx in txs, add (tx, r) to TxPool.
- Set h_{-1} to be the reference of chain$[-1]$ and h' to be the reference of chain$[-T]$ if $|\text{chain}| > T$ otherwise set h' to be the reference of chain$[0]$. Suppose that chain$[-1]$ belongs to modular chain j.
- Let txincludes be the transactions in TxPool that are not any block in modular chain $(j + 1 \bmod T)$, ordered by increasing round number.
- For $i = 1$ to q:
  - Pick a random nonce η and let $h = \mathsf{H}(h_{-1}, h', \text{txincludes}, \eta)$.
  - If $h < D_p$, then set chain := chain $\| (h_{-1}, h', \text{txincludes}, \eta, h)$, broadcast chain and break.
- Let $\mathcal{L} = [\text{List}_1, \ldots, \text{List}_T] = \text{modularize}(\text{chain}[: -T], T)$
- Output $\mathcal{L}$ to $\mathcal{Z}$. The final linearly ordered log output is taken as linearize$(\mathcal{L})$.

---

linearize$(\mathcal{L})$

- Output Aequitas$(\mathcal{L})$.

Figure 5.3: Protocol $\Pi^{\kappa,T}_{\text{mod}}$

choose R such that, except with negligible probability, R is larger than the time it takes for a transaction tx to be confirmed in all modular chains. It turns out that choosing R to be larger than the time to mine $5T/2$ blocks suffices.

Compliant parameters. We define a compliance predicate for $\Pi_{\text{mod}}(p)$ as $\Gamma^p_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$ if n, Δ, T, R are polynomials in κ; the parameters β, γ, τ are constants such that $\gamma - (\beta + \tau) > \frac{2}{3}$; and for all κ, $\Gamma^p_{\text{nak}}(n, \beta, \Delta) = 1$, $\Pi_{\text{nak}}(p)$ satisfies T-common-prefix and $(T, \mu > \frac{1}{2})$-chain-quality, $R \geq \frac{5T+2}{2g_0}$ where g_0 is such that $\Pi_{\text{nak}}(p)$ also satisfies $(T, g_0, \cdot)$-chain-growth.

Theorem 5.6. *Consider parameters such that* $\Gamma^p_{\text{mod}}(n, \beta, \Delta = 1, \gamma, T, \tau, R) =$

1. *Then,* $\Pi_{\text{mod}}^{\kappa,T}(p)$ *satisfies consistency,* R*-liveness, and* γ-batch-order-fairness *for* $(n, \beta, \Delta, \tau, R)$ *environments.*

We provide an overview of the proof technique in Section 5.4 and defer the full proofs to Appendix C.3.1. In Appendix 5.3.3, we also describe a variant that can tolerate any minority corruption using Fruitchains [204] as the underlying chain. We now state the main result for Π_{mod}.

5.3.2 Multi-Chain Protocol

We now introduce the protocol $\Pi_{\text{multi}}^{\kappa,d}$ (where d is the parallelization parameter), that uses multiple parallel Nakamoto chains.

Overview. Intuitively, in $\Pi_{\text{multi}}^{\kappa,d}$, honest nodes will mine d PoW chains in parallel, with each of the parallel chains serving as a semantic chain (i.e., an ordering for transactions). Similar to the single-chain protocol, $\mathsf{Aequitas}(\cdot)$ will now be run on the confirmed parts of each of the d chains. For simplicity, we will analyze the protocol only for $\gamma = 1$ and $\Delta = 1$. Following [206], we also use $p = \Theta(1/n)$ as the hardness parameter to make the per round mining rate $f = pqn$, be $\Theta(1)$ in κ. Figure 5.5 details our protocol and Figure 5.4 contains a simplified pictorial representation.

d-for-1 PoW mining. To mine blocks, we use a d-for-1 mining technique, similar to the one from [22], which generalizes the 2-for-1 trick from [129, 204]. Abstractly, nodes will simultaneously mine transactions on all d chains, and the chain to which the block gets appended to will depend on the hash value.

For this, we will slightly abuse the notation for blocks from Section 5.1.3. Specifically, a block will be a tuple $(\mathbf{h}_{-1}, \mathbf{h}', \mathbf{m}, \eta, h)$ where $\mathbf{h}_{-1}$ will be a list of parent blocks on all d chains, $\mathbf{h}'$ will be a list of reference blocks on all d chains, and $\mathbf{m}$ is the list of transactions to be (potentially) appended to each chain. This is called a *superblock* in [22] but for simplicity, we use the same formalism of a block. For our purpose, we will have $\mathbf{h}_{-1} = \mathbf{h}'$. We will use a random oracle H that outputs $d\kappa$ bits (instead of κ). For hardness parameter $p(\kappa)$, define $D_p = p(\kappa) \cdot 2^{\kappa}$. For a block with hash $h = \mathsf{H}(\mathbf{h}_{-1}, \mathbf{h}', \mathbf{m}, \eta)$, we will say that it is "mined into chain i", if $h(i) < D_p$ where $h(i)$ is the substring of h in range $(i-1)\kappa$ to $i\kappa - 1$. Considering different output bits for different chains allows us to make the mining for each chain independent although it does increase the size of the random oracle output. A block can be propagated to the rest of the network as part of the chain it gets mined in. For simplicity, we assume that the entire block is propagated, but note that in practice, similar to [22, 204], only the data relevant to its chain can be included, along with a commitment to the other discarded data.

Compliant parameters. We define a compliance predicate $\Gamma^{p}_{\text{multi}}(n, \beta, \Delta = 1, \gamma = 1, \tau, R, d, W) = 1$ if n, R, W are polynomials in κ with $p = \Theta(1/n)$; β, τ are constants such that $\beta + \tau < \frac{7-\sqrt{17}}{16}$; and for all κ, $\Gamma^{p}_{\text{nak}}(n, \beta, \Delta) = 1$, and there exists $T = T(\kappa)$ such that $\Pi_{\text{nak}}(p)$ satisfies T-common-prefix and $(T, \mu > \frac{1}{2})$-chain-quality, and $R(\kappa) \geq \frac{3T+2}{2g_0}$ where g_0 is such that $\Pi_{\text{nak}}(p)$ also satisfies $(T, g_0, \cdot)$-chain-growth. Furthermore, $W = \Theta(\kappa)$ and $d = \Theta(\kappa)$.

We will require R to be longer than the time it takes to mine $\frac{3T}{2}$ blocks in each chain. This will allow for tx to be mined and confirmed in all chains. W denotes the warmup time of our protocol.

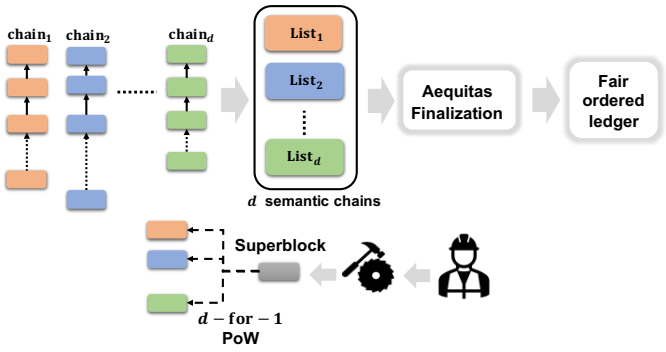

Figure 5.4: Pictorial representation of $\Pi^{\kappa,d}_{\text{multi}}$

We can now state the main result for Π_{multi}.

Theorem 5.7. *Consider parameters such that* $\Gamma^{p}_{\text{multi}}(n, \beta, \Delta = 1, \gamma = 1, \tau, R, d, W) = 1$. *Then,* $\Pi^{\kappa,d}_{\text{multi}}$ *satisfies consistency,* (W, R)*-liveness, and* 1-batch-order-fairness *for* $(n, \beta, \Delta, \tau, R)$ *environments.*

5.3.3 Fruitchains version of Π_{mod}

We briefly sketch a version of Π_{mod} that uses the Fruitchains protocol from [204] as the underlying protocol instead of Π_{nak}.

Background on Fruitchains. In the Fruitchains protocol, nodes create two types of objects: fruits and blocks. Fruits are the ones that actually contain transactions while blocks are used to order fruits (and thereby order transactions). Fruits are required to "hang" from recent blocks, where recency is defined using a recency parameter Y. To mine both fruits and blocks, nodes use the same PoW mining process (using the 2-for-1 mining technique [129]) which using a random oracle with output size 2κ. A block is said to be mined if the first κ bits of the hash are less than the mining hardness threshold D_p, while a

$\Pi^{\kappa,d}_{\text{multi}}(p)$

Initialize: $\mathcal{C} = [\mathsf{chain}_1, \ldots, \mathsf{chain}_d] = [\mathit{genesis}_1, \ldots, \mathit{genesis}_d]$; $\mathsf{TxPool} = \emptyset$; $T = T(\kappa)$ which is $\omega(\log \kappa)$

Upon receiving a valid $\mathcal{C}' = [\mathsf{chain}_1, \ldots, \mathsf{chain}_d]$:

- For all indices j, if $\left|\mathsf{chain}'_j\right| > |\mathsf{chain}_j|$, then set $\mathsf{chain}_j := \mathsf{chain}'_j$ and output any newly seen transactions to $\mathcal{Z}$.

In round r, upon receiving txs from $\mathcal{Z}$:

- For each tx in txs, add (tx, r) to TxPool.
- For all indices $j \in \{1, \ldots, d\}$:
  - Set both h^j_{-1} and h'_j to be the reference of $\mathsf{chain}_j[-1]$.
  - Let $\mathsf{txincludes}_j$ be the transactions in TxPool that are not any block in chain chain_j, ordered by increasing round number.
- Define $\mathbf{h}_{-1} = [h^1_{-1}, \ldots, h^d_{-1}]$, $\mathbf{h}' = [h'_1, \ldots, h'_d]$, and $\mathbf{txincludes} = [\mathsf{txincludes}_1, \ldots, \mathsf{txincludes}_d]$.
- For $i = 1$ to q:
  - Pick a random nonce η and let $h = \mathsf{H}(\mathbf{h}_{-1}, \mathbf{h}', \mathbf{txincludes}, \eta)$.
  - If for some j, $h(j) < D_p$, then set $\mathsf{chain}_j := \mathsf{chain}_j \parallel (\mathbf{h}_{-1}, \mathbf{h}', \mathbf{txincludes}, \eta)$
  - If a nonce solution was found, then broadcast $\mathcal{C}$ and break.
- Let $\mathcal{L} = [\mathsf{List}_1, \ldots, \mathsf{List}_d] = \mathsf{extract}(\mathcal{C})$
- Output $\mathcal{L}$ to $\mathcal{Z}$. The final linearly ordered log output is taken as $\mathsf{linearize}(\mathcal{L})$.

$\mathsf{extract}(\mathcal{C} = [\mathsf{chain}_1, \ldots, \mathsf{chain}_d])$

- Initialize all $\mathcal{L}_j$ to be empty.
- For all index $j \in \{1, \ldots, d\}$, and then for all $l < |\mathsf{chain}_j| - T$
  - Suppose that $\mathsf{chain}_j[l] = (\mathbf{h}_{-1}, \mathbf{h}', \mathbf{txincludes}, \eta, h)$. Append $\mathsf{txincludes}_j$ to $\mathcal{L}_j$.

$\mathsf{linearize}(\mathcal{L})$

- Output $\mathsf{Aequitas}(\mathcal{L})$.

Figure 5.5: Protocol $\Pi^{\kappa,d}_{\text{multi}}$

fruit is said to be mined if the last κ bits of the hash are less than some other threshold D_f. A block will be propagated to the rest of the network as part of the chain it gets mined in, whereas a fruit can be simply broadcast. The recency parameter mandates only recent fruits—those that hang from one of the last $Y\kappa$

blocks—are mined into a block.

Fair ordering protocol description. Consider the protocol $\Pi_{\text{fruit}}^{\kappa,T,Y}$ that mines a single fruitchain with recency parameter Y. T is the modular parameter for the protocol where $T = \Theta(\kappa)$ will be based on the consistency parameter for the underlying fruitchain. For a given block $\mathsf{chain}[b]$, the miner of the block indexes the fruits starting from one more than the index of the last fruit in the parent block ($\mathsf{chain}[b-1]$). Fruits in the same index modulo T will be considered to be semantically or logically connected. For example, fruits at indices 1, $T+1$, $2T+1$, and so on will be part of the same "modular chain." This divides the single PoW chain into T modular chains. We modularize the chains using fruits instead of blocks since the chain-quality property for Fruitchains is given in terms of fruits.

Now, in the same structure as Π_{mod}, from the T modular chains, T separate transaction orderings will be deduced which will be used as input for the $\mathsf{Aequitas}(\cdot)$ algorithm to retrieve the final transaction ordering. $\mathsf{Aequitas}(\cdot)$ will be run on the confirmed parts of the fruitchain i.e., with the last T blocks cut off, where $T = \Theta(\kappa)$. This will extract the fair ordering from the fruitchain. Note that the consistency parameter here is larger than the $\omega(\log \kappa)$ requirement for the standard Π_{mod}.

5.4 Proof Sketches

5.4.1 Overview of Proof Technique

We overview our key techniques for proving order-fairness and liveness for our protocols Π_{mod} and Π_{multi}. We note that the consistency of our protocols will directly follow from the consistency of the underlying Nakamoto chain.

Order-Fairness. Recall that in our protocols, honest nodes output the semantic chains to $\mathcal{Z}$ in an append-only fashion, and that the $\mathsf{linearize}(\cdot)$ function (which decides the final ledger) is defined as $\mathsf{Aequitas}(\cdot)$ (which solves the (weak) finalization problem) on the semantic chains. Therefore, it suffices to show dependency-completeness in order to use Lemma 5.5 and conclude batch-order-fairness.

Liveness. For liveness, we start with the following faithfulness property. Informally, a protocol is *dependency-faithful* if some honest node has received tx before tx′ whenever an edge from tx to tx′ exists in the dependency graph of an honest node.

Property 5.8 (Faithfulness of Dependencies)**.** Consider $(n, \beta, \Delta, \tau, R)$. We say that Π is *dependency-faithful* if for any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$-permissionless execution, the following holds: For honest node N in any round r, if N outputs $\mathcal{L}$ with $\mathsf{tx}' \lhd^{\Lambda} \mathsf{tx}$ where $\Lambda = (\Upsilon = (|\mathcal{L}|, \lceil \frac{|\mathcal{L}|}{2} \rceil - 1, 1), \mathcal{L})$, then there is at least one honest node that has received tx before tx′ as input from $\mathcal{Z}$. In other words, if an edge $(\mathsf{tx}, \mathsf{tx}')$ exists in $\mathsf{Graph}_{\Upsilon}(\mathcal{L})$, then some honest node must have received tx before tx′.

Now, through Lemma 5.9 (full proof in Appendix C.3), we show that if a protocol is dependency-faithful, then when $\Delta = 1$, if transactions tx and tx′ (even adversarial) are in the same Condorcet cycle, then they cannot have been received far apart. In other words, a transaction does not have to wait for an arbitrary length of time to ensure that it does not get combined with a later transaction in the same strongly connected component of the dependency graph. Consequently, this will enable transactions to be delivered in a bounded length of time satisfying liveness. Note that the specific liveness bound will be dependent on the actual protocol.

Lemma 5.9. *Consider* $(n, \beta, \Delta = 1, \tau, R)$ *and suppose that a protocol* Π *is dependency-faithful (Property 5.8). Suppose that some honest node in round* r *outputs* $\mathcal{L}$ *where* tx *and* tx′ *are in the same Condorcet cycle. Let* r, r' *be the first round that some honest node has received* tx *and* tx′ *respectively. Then* $r = r'$.

Corollary 5.9.1 (Informal)**.** *Consider a protocol* Π *that is dependency-faithful (Property 5.8) and suppose that* $\mathsf{Aequitas}(\cdot)$ *is used as the* $\mathsf{linearize}(\cdot)$ *function. If* $\Delta = 1$, *then* Π *will satisfy liveness (for some parameter).*

To summarize, it will be sufficient to show that are protocols are both dependency-completeness and dependency-liveness.

Remark 4 (Remark on Liveness.)**.** The Aequitas protocol from KZGJ achieves (conventional) liveness when no cycles can exist in the dependency graph or if the cycles cannot extend for across some bounded time interval. One such setting is when the external network (in the permissioned setting) has $\Delta_{\mathsf{ext}} = 1$. When reducing the problem in the permissionless setting to running $\mathsf{Aequitas}(\cdot)$ on the d lists, we emphasize that this no longer corresponds to requiring $\Delta = 1$ in our permissionless setting. Instead, this is the assumption that the sequence

number of any given transaction in all d lists differs by at most 1. In our protocols, this turns out to be false due to potential adversarial influence on all d input lists. Fortunately, when $\Delta = 1$ (in the permissionless environment), we can still show that any cycle that exists in the dependency graph of the d lists cannot extend for too long (Lemma 5.9). This allows our protocols to achieve conventional liveness.

5.4.2 Proof Sketch for Π_{mod}

We first note that consistency of $\Pi_{\text{mod}}^{\kappa,T}$ comes for free (when $\beta < \frac{1}{2}$) from the consistency property of Nakamoto's blockchain since we use a single PoW chain. For batch-order-fairness and liveness, we show that $\Pi_{\text{mod}}^{\kappa,T}$ is both dependency-complete and dependency-faithful. This is a relatively straightforward consequence of the chain-quality of the underlying Π_{nak}. Here, we sketch the proof for $\gamma = 1$, and defer the general result and full proof to Appendix C.3.1.

Completeness proof sketch. Consider $(\mathsf{tx}, \mathsf{tx}')$ such that all honest nodes received tx before tx'. Let $\mathsf{chain}[b]$ be the first block that contains either tx or tx'. Note that all blocks in $\mathsf{chain}[b : b + T - 1]$ will also contain at least one of tx and tx', thereby fixing the ordering between them in all semantic chains. Now, by chain-quality of the underlying Π_{nak}, more than half of the the blocks in $\mathsf{chain}[b : b + T - 1]$ are mined by honest nodes, i.e., will include tx earlier. Consequently, any honest dependency graph will contain the edge $(\mathsf{tx}, \mathsf{tx}')$.

Faithfulness proof sketch. Suppose that an honest dependency graph contains the edge $(\mathsf{tx}, \mathsf{tx}')$ (i.e., $\mathsf{tx}' \lhd \mathsf{tx}$ holds). Let $\mathsf{chain}[b]$ be the first block that

contains either tx or tx′; the blocks in $\mathsf{chain}[b : b + T - 1]$ then order the transactions in all semantic chains. Now, more than half of the semantic chains contain tx earlier but by chain-quality of the underlying Π_{nak}, less than half of them can be adversarial. Therefore, at least one honest node has mined a block with tx ordered earlier.

5.4.3 Proofs Sketch for Π_{multi}

Once again, consistency comes for free (when $\beta < \frac{1}{2}$) from the consistency of the underlying PoW chains. To prove order-fairness and liveness, we first define a cross-chain-quality as Property 5.10, and show that it implies both dependency-completeness and dependency-faithfulness for Π_{multi}. Finally, we show that Π_{multi} satisfies cross-chain-quality. The proofs for Π_{multi} are arguably more involved than those for Π_{mod} and we provide only a brief sketch here. The full proofs are deferred to Appendices C.3.2 and C.4.

Property 5.10 (Cross-Chain-Quality)**.** For a given view and tx, let $\mathcal{B}_{\text{tx}}$ be the set of blocks that contain tx for the first time in the confirmed part (i.e. $\mathsf{chain}_j[: -T]$) of each of the d chains. Then, more than half of the blocks in $\mathcal{B}_{\text{tx}}$ were mined by honest nodes.

Cross-chain-quality implies dependency-completeness and dependency-faithfulness (Lemma C.7). We prove this by contradiction. For any tx and tx′, we consider all the possibilities for how the transactions are mined in a given chain (e.g., by an honest or adversarial node) and how they were ordered in that chain (e.g., tx before tx′, tx after tx′, or at the same time). Then, we use a simple combinatorial argument to show that if dependency-completeness or

dependency-faithfulness were not satisfied, then either tx or tx′ would be need to be mined by the adversary in at least half of the parallel chains which contradicts cross-chain-quality.

Intuitively, this proof can be thought of as a generalization of the idea used to prove completeness and faithfulness of Π_{mod}; specifically, cross-chain-quality now plays an analogous role to the chain-quality property of Π_{nak} used earlier in the proofs for Π_{mod}.

Cross-chain-quality proof (Appendix C.4) The bulk of the technical novelty lies in showing that Π_{multi} satisfies the cross-chain-quality property. Roughly, we employ the following strategy. First, for a chain i and round r, we construct an event $V_i[r]$ such that whenever $V_i[r]$ holds, any transaction that is first seen by honest nodes at round r will be mined into an honest block. Next, we show that for a sufficiently small hardness parameter, the probability that this event happens, i.e., the transaction is mined into an honest block is more than 0.5. This means that by setting the number of chains to be large enough, we can now use the Chernoff bound to conclude that the transaction is contained in honestly mined blocks in more than half the chains, except with negligible probability. Equivalently, the cross-chain-quality property will be satisfied, except with negligible probability. Our proof makes novel use of techniques from the queuing theory literature and may be of independent interest. We point the reader to [115, 147] for a useful primer on the techniques we use.

To bound the probability for the event $V_i[r]$, we break it into two independent events $V_i^1[r]$ and $V_i^2[r]$ as follows:

- $V_i^1[r]$ is the event that in round r, there is no private adversarial chain that

is longer than the chain i.

- $V_i^2[r]$ is the event that the first block mined in chain i after round r is honest and the block persists in the chain forever (except with negligible probability).

At a high level, under the event $V_i[r]$, if a transaction is first seen in round r, then it will be contained in the first honest block mined after round r in chain i and it will persist. Now, we use techniques from queuing theory to compute the probabilities for each event. We highlight the non-triviality of computing the probabilities—primarily due to the possibility of adversarially propagated transactions. These are transactions that the adversary generates but does not immediately broadcast, instead attempting to first mine them into several parallel PoW chains. Abstractly, we need to carefully consider the scenario where an adversary privately mines a transaction onto several chains before releasing all the mined blocks at once, thus giving it a head start before honest nodes even see this transaction. We need to prove that even with such a head start, in more that half of the parallel chains, the transaction will be contained in an honestly mined block.

To compute the probability for $V_i^1[r]$, we first formulate the event as a discrete-time Markov chain (DTMC) and then show it to be positive recurrent and irreducible. We do this in order to use the ergodic theorem [147] and relate $P(V_i^1[r])$ to the stationary distribution of the DTMC. Finally, we use z-transforms to compute the stationary distribution probability. For $V_i^2[r]$, we first model the event as a one-dimensional random-walk which counts the difference between the number of honest blocks and the number of adversarial blocks mined in a given period. $P(V_i^2[r])$ can now be formulated as the prob-

ability that the random walk starting from an initial positive state never goes to zero (after a small warmup period). We use z-transforms to compute the random-walk probabilities.

5.4.4 Proof Sketch for Π_{fruit}

The proofs for security will almost exactly mirror the proofs for Π_{mod} and therefore, we only provide an outline here.

Pass and Shi [204] showed that there is some recency parameter $Y = \Theta(1)$ and some $\kappa_f = \Theta(\kappa)$ such that the Fruitchains protocol satisfies κ_f-prefix-consistency, and $(\frac{5\kappa_f}{\delta}, \mu = (1-\delta)(1-\beta))$-chain-quality for any $0 < \delta < 1$. For $\beta < \frac{1}{2}$, there exists a $\delta = \delta_0 > 0$ such that $\mu > \frac{1}{2}$. This means that for $T = \frac{5\kappa_f}{\delta_0}$, Fruitchains satisfies both T-consistency and $(T, \mu > \frac{1}{2})$-chain-quality. Note that this T is $\Theta(\kappa)$.

Now, exactly the same proofs for Π_{mod} can be used to show that $\Pi_{\text{fruit}}^{\kappa,T,Y}$ is both dependency-complete and dependency-faithful when $\gamma - (\beta + \tau) > \frac{1}{2}$ which will directly imply order-fairness and liveness. Note that consistency once again comes for free from the consistency of the underlying fruitchain.

5.5 Applications

We discussed previously how order-fairness is relevant for decentralized finance applications. Here, we provide additional potential applications, specifically in permissionless settings.

5.5.1 Zero-Block Confirmation

We show how fair ordering protocols can confirm non-conflicting transactions without mining a single block in the synchronous setting. We start by defining a *soft-ordering* property.

Definition 5.11 (δ-Soft-Ordering)**.** A protocol Π satisfies δ-soft-confirmation if the following property holds: If, an honest node receives tx in round r and tx′ in a round greater than $r + \delta$, then the final log contains tx before tx′.

Now, define $\mathsf{conflict}(\mathsf{tx}, \mathsf{tx}') = \mathsf{conflict}(\mathsf{tx}', \mathsf{tx}) = 1$ if tx and tx′ conflict with each other according to some semantic (e.g., spend the same tokens in a UTXO system). If two conflicting transactions are present in the output chain, only the first one will be executed by honest nodes to determine the current system state. Suppose that a node N receives tx in round r and is honest till at least round $r + \delta$. If N does not receive any tx′ such that $\mathsf{conflict}(\mathsf{tx}, \mathsf{tx}') = 1$ till round $r + \delta$, then all honest nodes will output tx before any conflicting tx′, and thus tx will be confirmed in the finalized ledger (due to the δ-soft-ordering property). This gives rise to a primitive that can be used to soft-confirm transactions. If δ is small, no blocks containing tx will be mined by this time, yet N will be able to soft-confirm tx, i.e., we get zero-block confirmation. Finally, we note that a transaction submitted by an honest node will not have any conflicting transactions at a future time, and thus can be confirmed by any other honest node within δ rounds.

It is easy to see that receive-order-fair protocols will satisfy 2Δ-soft-ordering since transactions received more than 2Δ apart will always be such that all nodes receive the first transaction before the second. Consequently, they also

provide zero-block-confirmation. Note that the probability that a soft-confirmed transaction is not valid in the finalized ledger is the same as the probability that the protocol does not satisfy order-fairness, which is negligible in κ.

Theorem 5.12. *If Π satisfies γ-receive-order-fairness w.r.t. $(\mathcal{A}, \mathcal{Z})$ with network parameter Δ, then it also satisfies (2Δ)-soft-ordering.*

Proof. Deferred to Appendix C.5. □

Remark 5. Although our protocols satisfy batch-order-fairness, and not receive-order-fairness, when $\Delta = 1$, they will satisfy (2Δ)-soft-ordering since any Condorcet cycle cannot extend past this time (Lemma 5.9), and therefore we still get zero-block-confirmation.

Soft ordering as Hamiltonian paths. While the previous application of soft-ordering allows fast confirmation of non-conflicting transactions, we provide another angle in terms of the possible orderings the final log could contain. For this, we relate possible final orderings to Hamiltonian paths in a graph built only using a node's local ordering. Details are given in Section 5.5.2.

Network considerations. While soft-ordering is an easy corollary of order-fairness, we emphasize that is not a direct artifact of network synchrony in a permissionless setting. Even when transactions are gossiped and reach all nodes within a synchronous Δ period, soft-confirmation within a small delay cannot be done if the underlying consensus protocol is not fairly ordered. For instance, suppose that a node N received tx in round r and a conflicting tx′ in round $r + 2\Delta + 1$, i.e. all nodes receive tx before tx′. Still N cannot soft-confirm

tx, since a miner proposing a block with tx′ first may not be rejected by the network (since in the view of a honest node who receives tx in round $r + \Delta$ and tx′ in round $r + \Delta + 1$, it is plausible that other nodes have received tx′ first.)

Larger soft-confirm values (small enough to still be zero-block) do not work either. Even if it is common knowledge for the current nodes that everyone received tx before tx′, the rejection of an adversarial miner's block with tx′ first, will require this additional information. This means that the determination of whether a chain is valid is no longer only dependent on the transactions in it. Consequently, when new nodes enter the system, they will not be able to determine the correct honest chain to mine on.

Another concern is nodes not gossiping double-spends as a mechanism to prevent DoS attacks. Fortunately, gossiping only the first instance of double-spend for a token is sufficient for an honest node to know not to locally soft-confirm a transaction; other double-spends for the same token need not be gossiped to the entire network. This allows us to retain our zero-block-confirmation property will still ensuring that flooding the network will not cause a DoS attack. It also does not affect honest transactions which are not double-spent.

5.5.2 Soft-Ordering as Hamiltonian paths

In the previous section, we discussed how the soft-ordering property allows honest transactions to be quickly locally confirmed when there are no conflicting or double-spend transactions. This requires a notion of "conflicting transaction" which may not always be clear. Here, we show another angle for looking at soft-confirmation in terms of the possible orderings the final log

could contain. In other words, we show how soft-confirmation allows nodes to narrow down possibilities for how the final ordered log might look, just based on their input orderings and long before the transactions are officially confirmed.

For this, suppose that a protocol Π satisfies δ-soft-ordering. Now, a node N constructs a graph $\mathcal{S}_N$ locally as follows, based only on the ordering in which it received transactions.

- For all transactions tx, a vertex labeled tx is added to the graph.
- For vertices tx and tx′ in $\mathcal{S}_N$ such that N received tx earlier, an edge (tx, tx') is added if N received tx more than δ rounds before tx′. On the other hand, both the edges (tx, tx') and (tx', tx) are added if tx was received less than δ rounds before tx′.

Now, consider the final totally ordered ledger $\mathbb{L}$ for the protocol Π. In case of transactions output at the same index, (as in batch-order-fairness), a suitable transformation is used to fully order these transactions. Recall that for batch-order-fairness, transactions are still totally ordered even inside a batch, but orderings within a batch do not account for unfairness.

We now show that at any point, there is some Hamiltonian path of $\mathcal{S}_N$ that is a prefix of the final ledger. Recall that a Hamiltonian path of a connected graph is a path that visits all vertices exactly once.

Lemma 5.13. *Consider a protocol Π that satisfies δ-soft-ordering. Let $\mathbb{L}_{\mathsf{view}}$ be the final totally ordered ledger for a view* view*. Let $\mathcal{S}_N^r$ denote the previously defined graph for*

node N at round r. Then for any round r, there is a Hamiltonian path P of $\mathcal{S}_N^r$ that satisfies the following property $\mathcal{Q}$:

- *Let V be the last vertex of P. Construct path P^* by removing from P all vertices that are reachable from V in $\mathcal{S}_N^r$. Informally, the removed transactions are the ones whose ordering is not yet fixed as their ordering could depend of transactions N receives shortly after.*
- *The transaction ordering given by P^* is a prefix of $\mathbb{L}_{\mathsf{view}}$.*

Proof. Deferred to Appendix C.5. □

CHAPTER 6

THEMIS: FAST, STRONG ORDER-FAIRNESS

This chapter introduces Themis, a practical order-fair consensus protocol, that mitigates several problems with prior order-fairness protocols.

The content in this chapter was initially published at ACM CCS 2023 with the title "Themis: Fast, Strong Order-Fairness in Byzantine Consensus" [158].

6.1 Themis Overview and Contributions

Themis operates in a partially synchronous, permissioned setting with n nodes of which $f < n/4$ may be arbitrarily adversarial. We implement Themis on top of Hotstuff [256], a popular leader-based protocol and show that it incurs minimal performance overhead.

Themis: Fair-ordering property. Themis achieves the *batch-order-fairness* property proposed by Kelkar et al. [161]. Informally, batch-order-fairness[1] (Definition 6.1) with parameter $\frac{1}{2} < \gamma \leq 1$ dictates that if γ fraction of nodes receive a transaction tx before tx′ from the client, then tx should be ordered *no later than* tx′. While [161] also introduced a stronger receive-order-fairness property where tx must be ordered strictly before tx′, it was shown to be impossible without strong synchrony assumptions.

Informally, *batches* arise as a result of non-transitive *Condorcet* cycles [88] in message receipt times across nodes (see Section 6.3 for more details); the relax-

[1]Here, *batch* is unrelated to the standard SMR optimization of increasing throughput (at the cost of latency) by amortizing consensus over many transactions.

ation is minimal in the sense that when there are no cycles, the stronger receive-order-fairness property can be satisfied.

We find however that these cycles can extend for arbitrarily long; this is ultimately responsible for the liveness problems with Aequitas. To get around this, we notice that in a typical deployment for Themis (e.g., for smart contracts), all transactions (even those in a batch) will require a total ordering for execution. Here, unlike Aequitas which can totally order transactions in a cycle only after all transactions in it are seen (which can take arbitrarily long), in Themis, using our technique of *unspooling* (see Section 6.3), we can output a batch part-by-part while still ensuring that all transactions in the same batch are output in an uninterrupted sequence. As a consequence, Themis imposes a total ordering on transactions, but one that respects batch-order-fairness.

Themis further guarantees another useful property: for any two consequent transactions tx_j and tx_{j+1} in the final output, it holds that tx_j was received before tx_{j+1} by at least $n(1-\gamma)+1$ honest nodes. This property also provides resistance against a particular kind of frontrunning attack—where the adversary wants to *immediately precede* a targeted user transaction.

We find that the fairness property supported in Themis (even for the weakest version, i.e., with $\gamma = 1$) is stronger in practice than other ordering notions proposed in Zhang et al. [259] and Kursawe [170]. This is showcased through our suite of fairness experiments.

Themis **design** (Section 6.4). Themis can be bootstrapped from any leader-based consensus protocol with minimal design changes: First, before constructing a block, all replicas send information about the order in which they received

client transactions to the current leader. Second, we specify an algorithm for an honest leader to construct a fair block proposal from the replica orderings. Finally, we provide a way for the replicas to verify the fairness of a leader's proposal as well as extract out the final ordering.

To construct a fair proposal, we extract out key techniques from [161] for ordering transactions (much as [157] does). Unfortunately, applying these techniques naïvely results in a loss of liveness, similar to the original Aequitas protocol from [161] which achieves only a weaker liveness notion due to the possible "chaining" of Condorcet cycles (see Section 6.3 for details). Concretely, in Themis, loss of liveness would mean empty blocks being produced by several honest leaders until the "chaining" of the current Condorcet cycle is completed.

Our solution is a new technique that we call *deferred ordering*. With deferred ordering, blocks produced by a leader contain some transactions that are fully ordered, while other transactions are only partially ordered. Partially ordered transactions await total ordering by a subsequent honest leader. Notably, the finalization of these partially ordered transactions *happens within the network delay* and does not have to wait indefinitely for the ordering of future transactions, e.g., the presence of Condorcet cycles. This feature of Themis allows us to circumvent the liveness problem of Aequitas. Thanks to deferred ordering, Themis achieves the standard liveness property. The technique is also of general interest: it can, e.g., be retrofitted to [161] to achieve standard liveness there too.

Theoretical SNARK-based design (Section 6.4.4). We also describe a more theoretical design—SNARK-Themis—which makes use SNARKs to achieve optimistic $\mathcal{O}(n)$ communication complexity. This is particularly notable since we

Protocol	Transaction Ordering	Comm. Complexity Optimistic	Comm. Complexity Worst	Corruption	Liveness	Censorship Resistance	Synchronized Clocks?
Aequitas [161]	γ-batch-order-fairness (Definition 6.1)	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	$n > \frac{4f}{2\gamma-1}$ (5)	Weak	Yes	No
Wendy [170]	Timed-Relative-Fairness(1) (Section 7.1)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$n \geq 3f+1$	Standard	Yes	Yes(2)
Pompē [259]	Ordering Linearizability(1) (Section 7.1)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$n \geq 3f+1$	Standard	No	Yes
Quick-Fairness [54]	κ-differential order-fairness(3)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$n \geq 3f+\kappa+1$	Only when all nodes are honest		No
Themis (This Work)	γ-batch-order-fairness(4)	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	$n > \frac{4f}{2\gamma-1}$ (5)	Standard	Yes	No
SNARK-Themis (This Work)		$\mathcal{O}(n)$	$\mathcal{O}(nf)$				

Table 6.1: Comparison of Themis to existing fair ordering protocols. An entry in **green** indicates that it is the best property among the protocols that we compare to. An entry in **red** indicates a significant shortcoming.
(1) *Fair separability* (Definition 7.1) captures both notions (see Section 7.1). Our experiments in Section 7 show why this notion is significantly weaker than the one achieved by Themis and is likely not enough to handle adversarial order-manipulation in real-world settings; **(2)** In Wendy, if honest local clocks are far apart (e.g., several seconds apart), then their fairness definition will never apply and therefore no fairness guarantees can be provided; **(3)** We show that κ-differential-order-fairness is simply a reparameterization of batch-order-fairness (see Section 9.8 and Appendix D.3); **(4)** Since we consider a total ordering in Themis, transactions within a batch are ordered contiguously for execution (see Section 6.3.3 and Section 6.4); **(5)** The fairness properties of both Aequitas and Themis are parameterized by γ but even for the weakest variant, i.e., corresponding to $\gamma = 1$, they are still stronger than those of Wendy [170] and Pompē [259]. For $\gamma = 1$, Aequitas and Themis both require $n \geq 4f + 1$ but also work with $n \geq 3f + 1$ if only crash-faults are present.

find that other fair-ordering protocols cannot easily be made optimistically $\mathcal{O}(n)$ even with the use of SNARKs. This makes SNARK-Themis the first such fair-ordering protocol; it's asymptotic communication complexity is in fact optimal and equivalent to state-of-the-art protocols that lack any fair ordering guarantees, e.g., Hotstuff [256].

Implementation and benchmarks (Section 6.5). We implement the $\mathcal{O}(n^2)$ version of Themis, and show that its integration with Hotstuff's codebase [174] results in small performance overhead in practice; our implementation with $n = 30$ nodes was still able to achieve a latency of roughly 53ms and a peak

throughput of $52,719$ transactions per second, which should suffice for most applications. Notably, Themis scales exactly as Hotstuff as n is increased. Furthermore, any overhead almost entirely vanishes when nodes are geo-distributed. Themis' source code along with instructions to reproduce our experimental results are available at: github.com/anonthemis/themis-src-anon. Since the public release of our paper, a prototype implementation of Themis has also been independently developed by Chainlink (the largest provider of oracle services for blockchain applications) as a practical emerging solution to the problem of order-manipulation [65].

Suite of fairness experiments (Section 7). There exists no prior work on practical measurement or empirical comparison of fair-ordering protocols. A key contribution of our work, therefore, is a systematically conceived suite of experiments to quantify the practical impact of both fairness definitions and protocols and understand their design tradeoffs. We consider both honest settings as well as broad classes of adversarial attacks common in practice. We study Themis through this lens and show that it provides significantly better fairness properties compared to other alternatives. We believe that our fairness suite will be useful for any future work on fair ordering protocols. We showcase three experiments:

1. *Ideal setting.* We quantify the strength of different fair ordering properties in an ideal honest setting (with no adversarial nodes) to understand the best-case scenario.

2. *Frontrunning and insertion attacks.* We evaluate resilience against network-layer insertions—attempts to maliciously insert transactions at the network layer (i.e., even before the consensus begins) through e.g., frontrunning.

We first prove that Themis does not allow any frontrunning under a natural assumption that the network respects triangle inequality. This result complements our experiments using a real network. As a concrete datapoint, for 100 geo-distributed nodes, for the fairness notions in [259, 170], up to 94% of the node connections in the network are still susceptible to adversarial frontrunning. In contrast, for the fairness notion in Themis, the number goes down to 2.8% in the simplest protocol parameter choice, and just 0.16% for the optimal parameter.

3. *Adversarial reordering.* We evaluate robustness to reordering attacks, i.e., attempts to maliciously reorder transactions compared to the honest execution. As a concrete datapoint, in setting with 101 nodes, it is easier to reorder in a median-timestamp based protocol (which essentially abstracts out the fairness components of [259, 170]) with just 5 adversarial nodes, than in Themis even given 25 nodes.

As a separate point, we also evaluate resistance to *censorship* (Section 7.6). In particular, we show a subtle censorship issue with Pompē [259], due to its use of an ordering phase prior to the actual consensus protocol.

6.2 Themis Model and Preliminaries

We now describe the basic model and preliminaries for Themis.

Model. Our setup is a permissioned system with a set $\mathcal{N}$ of n known protocol *nodes* or *replicas*, of which at most f are controlled by an adversary (denoted by $\mathcal{A}$) and can deviate arbitrarily from the protocol description. Transactions to be

sequenced are sent by system clients to all replicas. For our fair ordering protocols, we will consider the times at which a transaction was received by the replicas to decide on its overall ordering in the final ledger. For communication between replicas, we assume the presence of a PKI, and the security of digital signatures. The network itself is partially synchronous [105]; specifically, there exists a network delay Δ that bounds the message delivery time between replicas, but is not known to the replicas. $\mathcal{A}$ controls all message delivery, and can delay and reorder messages up to the bound Δ.

Graph terminology and algorithms. We make use of common graph algorithms to reason about the ordering dependencies between transactions. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ denote a graph with vertex set $\mathcal{V}$ and edge set $\mathcal{E}$. Unless specified, all graphs will be directed and unweighted. We often use vertices and transactions interchangeably when referring to graphs where vertices represent transactions.

$\mathcal{G}$ is a *tournament* graph if there is exactly one edge between each pair of vertices. For $V \in \mathcal{V}$, $\mathsf{SCC}_{\mathcal{G}}(V)$ denotes the strongly connected component that contains V. The subscript can be dropped when the context is clear. Recall that a strongly connected component (SCC) is a maximal subgraph such that there is a path in each direction between each pair of vertices in the component. $\mathcal{G}^*$ denotes the condensation of $\mathcal{G}$ (i.e., the transformed graph where vertices in the same SCC are collapsed into a single vertex). Note that $\mathcal{G}^*$ is guaranteed to be acyclic.

Within a graph, a Hamiltonian path is a path (i.e., a sequence of vertices) that visits each vertex exactly once. A Hamiltonian cycle is a Hamiltonian path that forms a cycle, i.e., there is also an edge from the last vertex to the first vertex

in the path. For an acyclic graph $\mathcal{G}$, a topological sorting is a linear ordering of vertices such that for any vertices U and V, U is ordered before V if $(U, V) \in \mathcal{G}.\mathcal{E}$.

Given a graph $\mathcal{G}$, its condensation can be easily computed in time $\mathcal{O}(|\mathcal{V}|+|\mathcal{E}|)$ using depth-first search techniques [237]. Moreover, if $\mathcal{G}$ is acyclic, then it can also be topologically sorted with the same asymptotic complexity. While the problem of detecting Hamiltonian paths in generic graphs is NP-complete, for acyclic graphs in particular, it is solvable in time $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ (through topological sorting). This is also true for tournament graphs [181].

6.3 Building Blocks

Our starting point is the fairness definitions of Kelkar et al. [161], and their Aequitas fair-ordering protocol. Our experiments show that their fairness property is stronger than the ones from other works, and therefore the focus of this work is to achieve the same strong property while fixing two problems with Aequitas: (1) the lack of standard liveness; and (2) the high communication complexity.

6.3.1 Aequitas Background

Batch-order-fairness. While Kelkar et al. [161] defined several fairness variants, the primary notion considered and subsequently realized by their leaderless Aequitas protocol, was batch-order-fairness, parameterized by $\frac{1}{2} < \gamma \leq 1$.

Definition 6.1 (γ-batch-order-fairness)**.** Suppose that transactions tx and tx′ are received by all nodes. If γn nodes received tx before tx′ locally, then all honest

nodes output tx *no later than* tx′.

A stronger notion, receive-order-fairness (exactly as Definition 6.1, but now, tx must be output *before* tx′), was also considered. An impossibility result, however, rules out realization of that notion except in specific synchronous settings. The impossibility result arises from the Condorcet paradox [88] in voting theory.

Abstractly, this allows for non-transitive global preferences even if each party's local preferences are transitive. As a simple example, suppose that three nodes receive transactions in the order $[a, b, c]$, $[b, c, a]$ and $[c, a, b]$. Here, each of "a before b", "b before c", and "c before a" holds for a majority of nodes, resulting in a non-transitive global preference.

Consequently, the global "fair" transaction ordering (according to receive-order-fairness) could contain cycles (even in a non-adversarial setting), which we call *Condorcet cycles* (we will often simply use the term cycle). Batch-order-fairness sidesteps this problem by enabling transactions to be output in *batches* (transactions within a batch can still be totally ordered at e.g., the application layer), and subsequently ignoring unfairness resulting from any cyclic orderings in the same batch. Importantly, **the batch relaxation is only used for transactions in the same cycle**.

Aequitas overview. The Aequitas protocol from [161] consists of three stages that each transaction goes through before being delivered to the ledger. First, in the *gossip* stage, all nodes broadcast transactions in the order they were locally received from clients. FiFo broadcast [148] is used to ensure that the broadcast order of an honest sender is maintained. Next, in the *agreement* stage, a variant of Byzantine Agreement [171] is used to agree on whose local orderings

to use to order a particular transaction. Lastly, the *finalization* stage is used to non-interactively determine the final transaction ordering. Given the local orderings of other nodes, the finalization algorithm builds a dependency graph of transactions as they arrive. Here, edges represent ordering dependencies between transactions (e.g., an edge from tx to tx′ signifies that a large number of nodes have received tx earlier). Disregarding the complexity introduced by the graph being built at different rates by different nodes, the core algorithm removes transactions from the graph and outputs them when, informally, they no longer have any dependencies.

Aequitas realizes batch-order-fairness and circumvents the receive-order-fairness impossibility by delivering transactions in the same Condorcet cycle at the same time (i.e., in the same "batch"). Further, Aequitas guarantees that batch-relaxation is minimal—if there are no Condorcet cycles, each batch includes only a single transaction (i.e., the stronger receive-order-fairness notion will be met).

6.3.2 Aequitas Technical Challenges

Weak-liveness. A crucial problem with the Aequitas protocol, however, is that it was proven to only guarantee a *weak* notion of liveness (given as Definition 6.3 for completeness). We find that once again, the presence of Condorcet cycles proves problematic here. In particular, we find that Condorcet cycles can "chain" together and form larger ones that can extend for arbitrarily long (in the worst case); informally, two cycles can be chained by having them share a transaction. This means that transactions input much later can become part of

the same cycle (although this can happen because of the specific input transaction orderings and not necessarily the adversary). In other words, specific input orderings could prevent transactions from being output for arbitrarily long.

Although Kelkar et al. [161] do not formally define these cycles, they are implicit in their weak-liveness definition. We show that, intuitively, an equivalent formulation of weak-liveness is that liveness for a transaction is guaranteed *only after its entire cycle is complete*, which may take arbitrarily long in the worst case (i.e., it cannot be a function of Δ). In contrast, (standard) liveness in partial synchrony needs to guarantee that transactions are output within a finite time dependent on the fixed (but unknown) parameter Δ.

Technical challenges and insights. Fixing the two aforementioned problems (weak-liveness and communication complexity) with Aequitas results in some technical challenges that required novel insights. Surprisingly, we find that the two problems are intertwined in a way that solving one actually requires solving the other.

We first prove that since Condorcet cycles can extend for arbitrarily long, if the protocol waits for all transactions in a cycle to be seen before they are output, it cannot achieve standard liveness; in fact, here, Aequitas' notion of weak-liveness is the best possible property.

To get around this, we notice that in a standard SMR deployment (e.g., for smart contracts), all transactions (even those in the same batch) will need to be *totally ordered* for execution. Note that this is compatible with batch-order-fairness and here, Aequitas will enforce any total ordering within a batch after it has been *fully seen*. Surprisingly, it turns out that designing the protocol from

the ground up with the final total ordering in mind allows us to get around the liveness problems. In so doing, we introduce *batch unspooling*, which intuitively allows for a batch to be output *part-by-part* (i.e., some transactions in a batch output before the rest) while still ensuring that all transactions in the batch are output *contiguously* (i.e., all transactions in a batch are output in an uninterrupted sequence). Towards this goal, our work needed to establish a **novel understanding of Condorcet cycles**.

Second, to reduce the communication cost, we use a common technique of routing communication through a *leader*. This turns out to be quite tricky however; a naïve design actually leads to higher cost. This happens because the presence of arbitrary-length Condorcet cycles can result in empty proposals even from honest leaders. To solve this, we introduce our technique of *deferred ordering* which enables the unspooling of a batch *across leaders*. In particular, deferred ordering allows for a leader proposal, in some cases, to contain partially ordered transactions which will be be totally ordered by a subsequent leader. Notably, we can still guarantee that the total ordering will be finalized *only based on the network delay* and therefore no weak-liveness problems arise.

6.3.3 Novel Understanding of Condorcet Cycles

While understanding cyclic ordering dependencies between transactions, we need to account for the fact that some replicas may be adversarial (and claim to have received transactions in a different order), as well as the fact that up to f honest nodes may not be considered (since we need to work in partial synchrony).

Condorcet cycles. We formally define (weak)-Condorcet cycles below as they lead to a key piece of our protocol design; these turn out to also be hidden within the weak-liveness definition of Kelkar et al. [161].

Definition 6.2 (Condorcet Cycle)**.** A list $[\mathsf{tx}_1, \ldots, \mathsf{tx}_l]$ is a (weak)-Condorcet cycle of length l if the following holds: For all $i \in \{1, \ldots, l\}$, where $\mathsf{tx}_1 = \mathsf{tx}_{l+1}$, at least $n(1-\gamma)+1$ **honest** nodes have received tx_{i+1} before tx_i.

Remark 6. While not important for our work, we can also define a stronger version based on the transaction ordering nodes "claim" to have received within the protocol. In particular, in the above definition, for each i, if at least $\gamma n - f$ nodes "claim" to have received tx_{i+1} before tx_i (honest nodes claim the order in which they receive transactions while adversarial nodes can claim arbitrary orderings), then $[\mathsf{tx}_1, \ldots, \mathsf{tx}_l]$ will become a *strong*-Condorcet cycle. The threshold $\gamma n - f$ is used here since it is guaranteed to hold when γn nodes receive tx_{i+1} before tx_i. Unless specified, we will use cycles from Definition 6.2.

Impossibility of liveness for entire batches. We start by showing a constructive example of an input transaction ordering that results in Condorcet cycles of *arbitrary length (and time)* for any parameters n, $f \geq 1$, and γ. This is done by continuously *chaining* together smaller Condorcet cycles to form larger ones. As an informal but illustrative example, a cycle $[\mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_l]$ can be formed through smaller cycles $[\mathsf{tx}_1, \mathsf{tx}_2, \mathsf{tx}_3]$, $[\mathsf{tx}_3, \mathsf{tx}_4, \mathsf{tx}_5]$ and so on where subsequent cycles are formed such that they share a common transaction. The overall construction is somewhat non-intuitive, but we give a general algorithm in Section 6.3.3.1 to explicitly build these arbitrary length cycles. In fact, we also show something stronger, namely the construction of arbitrary-length *strong*-Condorcet cycles.

As a consequence, if a protocol waits for the entire cycle to be seen before transactions within it are output (as Aequitas does), then standard liveness is impossible to achieve.

6.3.3.1 Condocet Cycle Chaining

We provide an explicit algorithm (Algorithm 1) for generating an arbitrarily large strong-Condorcet cycle for any valid n, f, γ (which also immediately implies the result for (weak)-cycles). For simplicity, we provide details for the synchronous setting where the "claimed" input orderings of all nodes are taken into account, but this can also be extended to only using $n - f$ orderings. Abstractly, the algorithm inductively constructs transaction input orderings for each node in a way that all transactions will be part of the same strong-cycle.

Suppose that the system nodes are denoted by the set $\{0, \ldots, n-1\}$. To create a strong-cycle of length $l \geq n$ (this is a non-optimal bound on the minimum length of a cycle with n nodes), the algorithm proceeds by iteratively adding a new transaction to all input lists while maintaining the invariant that all current transactions form a strong-cycle. For the initial transaction tx_0, the input ordering txlist_j of each node j is taken simply as $[\mathsf{tx}_0]$. To add a later transaction tx_i, a set S_i of size $\gamma n - f$ is first chosen. S_1 is chosen to be the set $\{0, \ldots, \gamma n - f - 1\}$. The set S_i is obtained by cyclically rotating S_{i-1} modulo n. For a node $j \in S_i$, tx_i is inserted right before before tx_{i-1} in txlist_j; for other nodes $j \notin S_i$, tx_i is appended to the end of txlist_j. By doing so, we make sure that tx_i occurs before tx_{i-1} in at least $\gamma n - f$ lists.

Additionally, it is easy to see why tx_0 will occur before tx_l in at least $\gamma n - f$

Algorithm 1: Explicit input ordering with unbounded Condorcet chaining for n, f, γ

Let $N = \{0, \ldots, n-1\}$ be the set of all nodes.;
Let $\mathsf{txlist}_0, \ldots, \mathsf{txlist}_{n-1} \leftarrow [\mathsf{tx}_0]$;
$\mathsf{start} \leftarrow 0$;
for $i \in 1, 2, \ldots$ **do**
    $\mathsf{end} \leftarrow (\mathsf{start} + (\gamma n - f) - 1) \mod n$
    if $\mathsf{start} < \mathsf{end}$ **then** $S = \{\mathsf{start}, \ldots, \mathsf{end}\}$;
    else $S = \{\mathsf{start}, \ldots, n-1\} \cup \{0, \ldots, \mathsf{end}\}$;
    for $j \in S$ **do** Insert tx_i right before tx_{i-1} in txlist_j;
    for $j \in N \setminus S$ **do** Append tx_i at the end of txlist_j;
    $\mathsf{start} \leftarrow (\mathsf{start} + 1) \mod n$
end

lists through induction. First note that tx_0 is before tx_1 in $n - (\gamma n - f) = n(1 - \gamma) + f$ lists. Further, if tx_d occurs after tx_0 in some list, then for any $d' \geq d$, $\mathsf{tx}_{d'}$ will also occur after tx_0 in that list (since each transaction is added just before the last one or at the end of the list). Now, due to the cyclic rotation of the S_i, each tx_j will be after tx_0 in one more list (subject to a maximum of n lists) than tx_{j-1} was. Consequently, for any γ, f, when $l \geq n$, tx_0 will occur before tx_l in at least $\gamma n - f$ lists. When $\gamma = 1$, $l = n$ will be the first iteration that this happens.

Finally, putting this together, we can conclude that $[\mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_l]$ forms a strong-Condorcet cycle.

Weak-liveness in terms of cycles. Looking at the weak-liveness definition from Kelkar et al. [161], it is easy to see that any tx′ part of the same cycle as tx will need to be added to T. This means that if the cycle extends for arbitrarily long, new transactions will keep being added to T, delaying the delivery of transactions in that cycle. The Aequitas protocol, therefore, can only guarantee liveness after the current cycle completes, which may take arbitrarily long. While the weak-liveness definition mentions one honest node, we note that Ae-

quitas achieves a stronger property (for a general γ) where $n(1-\gamma)+1$ is used as the threshold instead.

Definition 6.3 (Weak-liveness [161])**.** Consider tx received by all protocol nodes and define the set T built recursively as follows: (1) Add tx to T; (2) For any $m \in \mathsf{T}$, add to T all m' such that at least one honest node has received m' before m. At any point, if all honest nodes have received all transactions in T, then tx will eventually be delivered by all honest nodes.

6.3.3.2 Batch Unspooling

We bypass this impossibility using a new technique—*batch unspooling*. In typical consensus use cases (e.g., smart-contract settings), it is necessary to *totally order* all transactions for execution. For this, Aequitas supports enforcing any total ordering for transactions within a batch. Still, this is done only at the end *after* the entire cycle has been seen. The final ordering can now be thought of being linearly partitionable into cycles; in particular, $[\mathsf{tx}_1^{(1)}, \ldots, \mathsf{tx}_{l_1}^{(1)}, \mathsf{tx}_1^{(2)}, \ldots, \mathsf{tx}_{l_2}^{(2)}, \ldots]$ where $\mathsf{tx}_1^{(i)}, \ldots, \mathsf{tx}_{l_i}^{(i)}$ are part of the same cycle.

Perhaps surprisingly, we find that embedding the total ordering requirement *within the protocol design* itself allows us to provide (standard) liveness while still *keeping the same fairness guarantees*. We call this technique *batch unspooling*. Unspooling allows us to output transactions within a cycle *without waiting for the full cycle* to be seen. Nevertheless, we can guarantee that transactions within the same cycle will be output *contiguously*, i.e., no transaction from a later cycle will be ordered before all transactions from the current cycle are ordered. This allows us to **achieve the same fairness property** as Aequitas (in the scenario where final execution requires a total ordering) while providing standard liveness.

We note that figuring out when the current cycle ends brings back the same weak-liveness problem, but our technique of batch unspooling makes it so that this is no longer required.

6.4 Themis Description

We now describe our protocol Themis in detail. Themis can be bootstrapped from *any existing leader-based protocol* and will endow it with fair ordering with minimal design changes.

Overall design. Recall that standard leader-based protocols allow the leader to *unilaterally* choose its block proposal—only the validity of transactions and not their ordering is checked by the replicas. Themis will achieve fairness by providing a mechanism for replicas to also check the *ordering* in the proposal. To enable the leader to construct a fair ordering, all replicas will first submit their *local transaction orderings* to the leader. By local ordering, we mean transactions received at that replica ordered by their receive times.

Looking ahead, to enable batch unspooling across multiple leaders, we introduce our technique of *deffered ordering*. In line with this, we describe two algorithms to be executed by an honest leader: `FairPropose` for constructing (partial) proposals, and `FairUpdate` for any updates to previous proposals. In our core Themis design, the leader will send all the local orderings to all replicas to enable checking the fairness of its proposal. As a more theoretical alternative, we also later describe SNARK-Themis which leverages SNARKs to reduce the asymptotic complexity of this step.

Finally, we also describe an algorithm `FairFinalize`, that allows the replicas to extract a fair transaction ordering from fully specified proposals (we will elaborate on this later).

Local replica orderings. Before the leader constructs a proposal, each replica sends its local transaction ordering to the leader. Note that since the network is partially synchronous, the leader will need to work with only $n - f$ orderings. To enable the leader to construct a proposal, an honest replica i sends the following: (1) List_i containing the transactions received by i that are not part of any previous proposal, in the order that they were received (by i); (2) $\sigma_i(\mathsf{List}_i)$ which is i's signature on List_i. Further, to allow the leader to update previous proposals, i sends the following: (1) Update_i containing transactions from previous proposals that are not fully specified in the order that they were received (by i); (2) $\sigma_i(\mathsf{Update}_i)$ which is i's signature on Update_i. Notably, our protocol only requires *orderings* from the replicas, and not individual timestamps for transactions. In other words, Themis does not need to rely on synchronized clocks.

6.4.1 Constructing the Leader Proposal

The leader starts by checking the validity of the local replica orderings by checking their signatures. Then, to construct its proposal, the leader executes `FairPropose`($\mathcal{L}$), where $\mathcal{L}$ is a set of $n - f$ replica orderings. Abstractly, the goal of this algorithm is to propose as many transactions as possible while making sure that the exclusion of any transaction from the proposal does not violate fairness.

Replica ordering notation. For a set $\mathcal{L}$ of $n-f$ orderings, we use tx $\in_k \mathcal{L}$ (resp. tx $\notin_k \mathcal{L}$) to denote that tx is present in at least (resp. less than) k orderings in $\mathcal{L}$. We use tx $\prec_{(\mathcal{L},k)}$ tx′ to denote that tx appears before tx′ (this includes the case where only tx is present) in at least k orderings in $\mathcal{L}$. We use $\mathsf{Weight}_{\mathcal{L}}(\text{tx}, \text{tx}')$ to denote the maximum value k such that tx $\prec_{(\mathcal{L},k)}$ tx′.

Deferred ordering intuition. Abstractly, the final total transaction ordering can be thought of as being partitioned into cycles. While enabling batch unspooling across multiple leaders, we still need to guarantee that no transaction from a *later cycle* is output before all transactions from the current cycle are output. Towards this, based on the local replica orderings, the leader can split transactions into three types: *solid*, *blank*, and *shaded*. We use *non-blank* to label a transaction that is either solid or shaded. Very roughly, this delineation enables the following:

1. Solid transactions are the ones that are present in many replica orderings and can be sequenced within the current proposal. This property can be guaranteed for tx if tx $\in_{n-2f} \mathcal{L}$.

2. Blank transactions are present in too few replica orderings and can be *excluded* from the current proposal. In particular, we can guarantee that if tx is blank, then it *cannot be part of a previous cycle* than any transaction in the current proposal; i.e., excluding it will not result in any unfairness. This happens when tx $\notin_{n(1-\gamma)+f+1} \mathcal{L}$.

3. Shaded transactions take a more indeterminate form. These are transactions tx such that tx $\in_{n(1-\gamma)+f+1} \mathcal{L}$ but tx $\notin_{n-2f} \mathcal{L}$. In many cases, we can rule out the possibility of tx occurring in a previous cycle than all transactions in the

current proposal, in which case tx can safely be excluded. In some cases however, current information does not allow for this to be ruled out—here, we must include the shaded transaction within the current proposal.

Doing so can result in a somewhat different challenge. If two or more such shaded transactions are included, then the ordering *amongst them* may not be clear yet. Our deferred ordering technique applies exactly here. It allows proposing a partial ordering for shaded transactions which will be completed at a later point when they are present in $n - 2f$ replica orderings. Notably, this happens only based on the actual network delay for these transactions to be received, and is therefore independent of other transactions.

Leader proposal algorithm. We now detail the `FairPropose` algorithm. Since some transactions may be partially ordered, we will have `FairPropose` output a *dependency* graph $\mathcal{G}$ that contains ordering dependencies within transactions in the current proposal. Later, through `FairFinalize`, replicas will extract out the final fair ordering from this graph.

The dependency graph $\mathcal{G}$ is constructed as follows: First, a vertex is added for each non-blank transaction. An edge $(\mathsf{tx}, \mathsf{tx}')$ is added to the graph whenever $\mathsf{tx} \prec_{(\mathcal{L},\mathsf{thresh})} \mathsf{tx}'$ where $\mathsf{thresh} = n(1-\gamma) + f + 1$. Intuitively, this threshold is chosen so that an edge $(\mathsf{tx}, \mathsf{tx}')$ signifies that tx' *cannot be in an earlier cycle* than tx. Only one of $(\mathsf{tx}, \mathsf{tx}')$ or $(\mathsf{tx}', \mathsf{tx})$ is added however, i.e., if both conditions are satisfied, the one with the larger value will be added. Specifically, let $k = \mathsf{Weight}_{\mathcal{L}}(\mathsf{tx}, \mathsf{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}}(\mathsf{tx}', \mathsf{tx})$. If both $k, k' \geq \mathsf{thresh}$, then add the edge $(\mathsf{tx}, \mathsf{tx}')$ if $k > k'$ and the edge $(\mathsf{tx}', \mathsf{tx})$ if $k' > k$; If $k = k'$, then one of the two edges can be added deterministically. Note that no edges are added when both $k, k' < \mathsf{thresh}$. When this happens, an edge will be added by a later

Algorithm `FairPropose(`$\mathcal{L}$`)`

`// Propose a fair ordering for new transactions.`
On input a set $\mathcal{L}$ containing the local transaction orderings of $n - f$ nodes:

1. **Build Dependency Graph**
   - Create an empty graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.
   - For each non-blank tx, add a vertex tx to $\mathcal{V}$.
   - For each $\mathsf{tx}, \mathsf{tx}' \in \mathcal{V}$, let $k = \mathsf{Weight}_{\mathcal{L}}(\mathsf{tx}, \mathsf{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}}(\mathsf{tx}', \mathsf{tx})$. If $k \geq n(1-\gamma) + f + 1$ and $k \geq k'$ and $(\mathsf{tx}', \mathsf{tx}) \notin \mathcal{E}$, then add the edge $(\mathsf{tx}, \mathsf{tx}')$ to $\mathcal{E}$.
   - Compute the condensation graph $\mathcal{G}^*$ of $\mathcal{G}$ and its topological sorting S.
2. **Output Fair Ordering**
   - Let V be the last vertex of S that contains a solid transaction.
   - Remove those transactions from $\mathcal{G}$, that are in vertices after V in S.
   - Output $\mathcal{G}$.

Figure 6.1: Leader Proposal Algorithm

proposal.

Next, we need to figure out which shaded transactions need to be included in the proposal and which ones can be excluded (recall that all solid transactions will be proposed). In particular, the proposal will contain exactly those shaded transactions that have a path (in $\mathcal{G}$) to a solid transaction.

For this, we compute the condensation of $\mathcal{G}$ and topologically sort it. Let V be the last vertex in this sorting that contains a solid transaction. Then `FairPropose` outputs the resulting graph by removing all tx from $\mathcal{G}$ where $\mathsf{SCC}(\mathsf{tx})$ occurs after V in the topological sorting; everything after this needs to be excluded. Details are given in Figure 6.1.

Proposal properties. The output of `FairPropose`($\mathcal{L}$) is guaranteed to contain all solid transactions in $\mathcal{L}$ and no blank transactions. Further, it contains exactly those shaded transactions that contain an outgoing path into a solid transaction within the dependency graph $\mathcal{G}$. Further, there is exactly one edge between any two vertices, except when both vertices are shaded (in which case there might be none). We prove this graph structure in Lemma D.1. Future leaders cannot modify existing edges but will be able to add missing edges between shaded transactions. As mentioned before, our deferred ordering technique of proposing a "partial graph" is crucial as it allows us to avoid the weak-liveness problem.

To show that the graph proposal is consistent with batch-order-fairness, we need to show that any transaction exclusion does not violate fairness. In particular, we prove (in Lemma D.3) that any excluded transaction that was excluded *cannot be in a previous cycle* than any transaction that was included. This implies that cycles are output *contiguously*, i.e., all transactions from the current cycle will be output before any transaction from a subsequent cycles.

Update algorithm. The algorithm `FairUpdate` allows a leader to add missing edges to a previous leader's proposal graph. For this, each replica i also sends the ordering Update_i that orders shaded transactions from previous proposals as seen by i. The update algorithm is quite simple: upon receiving $\mathcal{L}_{\text{updates}}$ containing $n - f$ such replica orderings, for all transactions tx and tx′ that were part of the same previous leader's proposal and do not currently have an edge between them, the edge (tx, tx') is added if $\text{tx} \prec_{(\mathcal{L}, n(1-\gamma)+f+1)} \text{tx}'$. Once again, only one of (tx, tx') or (tx', tx) is added however exactly as in the `FairPropose` algorithm. The new edges are specified in an update list $\mathcal{E}_{\text{updates}}$. Intuitively, this

Algorithm `FairUpdate(`$\mathcal{L}_{\text{updates}}$`)`

`// Update the ordering for previous proposals`
On receiving a set $\mathcal{L}_{\text{updates}}$ containing the local transaction orderings of $n - f$ nodes for previously proposed shaded transactions:

1. **Output Dependencies**

   - Let $\mathcal{E}_{\text{updates}} \leftarrow \emptyset$.
   - For all tx and tx$'$ that are part of the same leader proposal and between whom no edge has been proposed yet, let $k = \mathsf{Weight}_{\mathcal{L}_{\text{updates}}}(\text{tx}, \text{tx}')$ and $k' = \mathsf{Weight}_{\mathcal{L}_{\text{updates}}}(\text{tx}', \text{tx})$. If tx $\in_{n-2f} \mathcal{L}_{\text{updates}}$, $k \geq k'$ and $k \geq n(1-\gamma)+f+1$ then add the edge (tx, tx') to $\mathcal{E}_{\text{updates}}$.
   - Output $\mathcal{E}_{\text{updates}}$.

Figure 6.2: Update Algorithm

allows all nodes to use these update edges to fill in the missing details within earlier proposals in order to compute the final fair ordering. We provide details in Figure 6.2.

The complete leader proposal can now be defined as the block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\mathcal{L}, \mathcal{L}_{\text{updates}}))$. Replicas will use π to verify fairness; in SNARK-Themis, this can be done with a SNARK.

Consensus integration. The consensus design changes required are minimal. To enable the leader to construct its proposal, all replicas will first submit their local transaction orderings. The leader will now construct its proposal B and use the underlying consensus protocol to confirm it. Within this, replicas will also verify the *fairness* of the proposal (using π) before providing their signature. Since π contains all the replica orderings, the communication cost of the protocol will be $\mathcal{O}(n^2)$; this can theoretically be reduced to $\mathcal{O}(n)$ (when the leader is honest) by making π a SNARK instead. As a standard optimization, only the proposal hash needs to be signed. Note that other parts of the underlying protocol,

Complete Themis Description

Round leader protocol:

1. Wait for $n - f$ correctly signed lists $(\mathsf{List}_i, \mathsf{Update}_i)$ from replicas. Let $\mathcal{L} = \{\mathsf{List}_i\}$ and $\mathcal{L}_{\text{updates}} = \{\mathsf{Update}_i\}$
2. Run `FairPropose`$(\mathcal{L})$ and `FairUpdate`$(\mathcal{L}_{\text{updates}})$ to get $\mathcal{G}$ and $\mathcal{E}_{\text{updates}}$ respectively.
3. Run the underlying Consensus protocol to agree on a valid $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\mathcal{L}, \mathcal{L}_{\text{updates}}))$ and append it to the list of agreed upon blocks.

Replica protocol (for N_i):

1. Let List_i denote the ordered list of transactions received by N_i that are not yet part of any leader proposal. Let Update_i denote the ordered list of transactions received by N_i that are correspond to vertices with missing edges in some previous leader proposal.
2. Send $\mathsf{List}_i, \mathsf{Update}_i$ to the current round leader and verify that the leader's response is a valid proposal (by checking the SNARK proofs).
3. Participate in the underlying Consensus protocol to agree on a block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}})$ and append it to the list of agreed upon blocks.

Constructing the final ordering (for N_i):
At any point, N_i can do the following to append new transactions to its final ordering:

1. Let B_j be the first block whose transactions have not yet been output in the final ordering by N_i, and let B_k be the final block.
2. Run `FairFinalize` with $B_j, \ldots, B_k$ to obtain an ordering $[\mathsf{tx}_p, \ldots, \mathsf{tx}_q]$ of transactions.
3. Append $\mathsf{tx}_p, \ldots, \mathsf{tx}_q$ to the final transaction ordering.

Figure 6.3: Complete Description of Themis

e.g., leader election and view-change will remain exactly the same. Figure 6.3 provides a complete description.

Replica verification. Replicas can easily verify the fairness of the block $B = (\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi)$. In particular, given all the orderings within $\pi = (\mathcal{L}, \mathcal{L}_{\text{updates}})$, replicas can simply run `FairPropose` and `FairUpdate` themselves to ensure that the proposed $\mathcal{G}$ and $\mathcal{E}_{\text{updates}}$ are both correct.

6.4.2 Finalizing the Fair Ordering

Fully specified proposals. Recall that leaders can propose partial graphs for which missing edges will be added by future leaders. A proposal is *fully specified* if its graph is a *tournament*, i.e., there is exactly one edge between each pair of vertices. An important point to highlight here is that the confirmation of transactions will depend only on the network delay and does not suffer from the weak-liveness problem present in Aequitas. This is because missing edges in $\mathcal{G}$ will be added (making it fully specified) as soon as the transactions are received by the network and an honest leader is elected.

Transactions in fully specified proposals can be totally ordered and output by Themis. The algorithm `FairFinalize` describes how honest replicas can extract this final ordering.

Finalization algorithm. Given a sequence of proposals $B_1, \ldots, B_l$ agreed upon by the underlying consensus protocol, whose transactions have not been fairly ordered yet, the first step is to add any missing edges; in particular, if tx and tx′ are transactions in $B_i.\mathcal{G}$ such that $(\text{tx}, \text{tx}') \in B_j.\mathcal{E}_{\text{updates}}$ for some $j > i$, then add the edge (tx, tx') to $B_i.\mathcal{G}$. After the updates, let k be the largest index such that all $B_i.\mathcal{G}$ $(i \leq k)$ are now fully specified. Intuitively, transactions in these blocks will be ordered by `FairFinalize`.

To order transactions in $B.\mathcal{G}$, we first compute its condensation $B.\mathcal{G}^*$ by collapsing its SCCs and then find the topological sorting $[V_1, V_2, \ldots, V_c]$. Note that each of V_1 to V_c represent an SCC within $B.\mathcal{G}$. To output the final total ordering, we need to now order transactions within each V_i. As an optimization,

transactions whose SCC as well as all SCCs that precede it contains only solid transactions can also be finalized immediately even before $B.\mathcal{G}$ is fully specified.

Observe that it is sufficient to order transactions in each V_i arbitrarily (since intuitively they represent cyclic dependencies) using an agreed-upon function; a similar technique is used in Aequitas. As a better alternative however, we show how to take advantage of useful hidden properties within the graph to provide stronger fairness guarantees, which we describe next.

Stronger fairness guarantees within SCCs. Instead of arbitrarily ordering transactions in an SCC, we will employ Hamiltonian cycles (a graph cycle that visits every vertex exactly once) to order transactions in a way that connects to the ordering in other SCCs.

An old result by Camion [56] shows that all strongly connected tournaments contain a Hamiltonian cycle. Note that although the problem of finding Hamiltonian cycles is NP-complete for general graphs, for tournaments specifically, they can be found in time linear in the number of edges [181].

Concretely, for each SCC C in $B.\mathcal{G}$, we first find a Hamiltonian cycle H_C containing all transactions in C. If multiple cycles exist, one of them can be chosen deterministically to be used for the total ordering. Now, the ordering for transactions within V_1 to V_{c-1} (of the topological sorting) is simply the sequence of the Hamiltonian cycles H_{V_1} to $H_{V_{c-1}}$. For the final SCC V_c, we will cyclically rotate the corresponding Hamiltonian cycle. Recall that our `FairPropose` algorithm guarantees that this contains at least one solid transaction, say tx. Now, to obtain the ordering for V_c, we rotate H_{V_c} so that tx is now the final transaction among those output.

Algorithm `FairFinalize()`

Given a sequence of proposals $[B_1, B_2, \ldots, B_l]$:

1. **Update Graphs**
   i. For all B_i and transactions tx, tx′ in B_i that do not have an edge between them, if (tx, tx') is in some $B_j.\mathcal{E}_{\text{updates}}$, then add that edge to $B_i.\mathcal{G}$.
   ii. Let k be the last index such that the graph $B_k.\mathcal{G}$ is a tournament.
   iii. Compute the condensation graphs of $B_1.\mathcal{G}, \ldots, B_k.\mathcal{G}$ and their topological sortings $S_1, \ldots, S_k$.
2. **Retrieve Fair Ordering**
   i. For each $S_i = [v_{i1}, \ldots, v_{il_i}]$ where $i \in \{1, \ldots, k\}$, let H_{ij} be a Hamiltonian cycle of v_{ij} (pick one deterministically if there are multiple, e.g., by removing the edge with the smallest weight). Note that this exists since each v_{ij} is a strongly connected tournament.
   ii. Let tx_{il_i} be a solid transaction in each v_{il_i} (pick one deterministically if there are multiple), and let H'_{il_i} be the cyclic rotation of H_{il_i} so that the last transaction is tx_{il_i}.
   iii. The final transaction ordering for the block B_i is now given by $H_{i1}, \ldots, H_{i(l_i-1)}, H'_{il_i}$.

Figure 6.4: FairFinalize Algorithm

The upshot is that we can now *cleanly connect* the boundaries of subsequent Condorcet cycles. Specifically, if $[\text{tx}_1, \ldots, \text{tx}_l]$ is the final output ordering, then for any j, tx_j is received before tx_{j+1} by at least $n(1-\gamma)+1$ honest nodes, regardless of whether tx_j and tx_{j+1} are in the same cycle. Notably, this is not achieved if transactions in a cycle are ordered e.g., alphabetically (as in Aequitas [161]).

A nice practical consequence of this property is strong resilience against a particular kind of front-running attack—specifically the one where the adversary wants to place its transaction *immediately before* an honest user's transaction.

6.4.3 Formal Themis Results

Consider a partially synchronous network with $n > \frac{4f}{2\gamma-1}$ nodes where f is the maximum number of adversarial nodes and $\frac{1}{2} < \gamma \leq 1$ is the order-fairness parameter. In this regime, we formally state the properties of Themis. Proofs are deferred to Appendix D.1.

Theorem 6.4 (Themis fairness)**.** *At any time, let* $[\mathsf{tx}_1, \ldots, \mathsf{tx}_l]$ *be the total transaction ordering output by* Themis*. Then,* Themis *satisfies:*

1. (γ-batch-order-fairness*). The ordering can be linearly partitioned in batches satisfying* batch-order-fairness*. Concretely, there are indices* $1 = i_1 < \cdots < i_k = l + 1$ *such that* $[C_1, \ldots, C_{k-1}]$*, where each* $C_j = \{\mathsf{tx}_{i_j}, \ldots, \mathsf{tx}_{i_{j+1}-1}\}$*, satisfies* γ-batch-order-fairness*.*

Importantly, (similar to Aequitas), the batches are minimal—they are of size one when there are no cyclic ordering dependencies. As a corrolary, receive-order-fairness is satisfied when there are no cycles.

2. (Consequent-transaction fairness). For each $i \in \{1, \ldots, l-1\}$*, at least* $n(1-\gamma)+1$ *honest nodes received* tx_i *before* tx_{i+1}*.*

Corollary 6.4.1. Themis *satisfies receive-order-fairness when there are no Condorcet cycles.*

We also show that Themis satisfies the standard SMR consistency and liveness properties.

Theorem 6.5. Themis *satisfies consistency and liveness.*

Audit friendliness. Themis' fairness properties are also quite friendly for auditing. In particular, an optimistic fast path can be used where the leader constructs the proposal as normal but does not send any correctness proof to the replicas; instead, the proposal can be validated at a later time, even by an external auditor. This is of course possible only when the auditor can force a revert of the system to an earlier state and/or impose a penalty upon detection of a malicious leader proposal.

Crash-fault tolerant protocol for $n \geq 3f + 1$. Themis can also be modified in a straightforward way to create an $3f + 1$ version (when $\gamma = 1$) in settings with only crash faults (instead of explicit adversarial behavior). Interestingly, the same protocol will work for Byzantine faults but at the cost of allowing the leader node to censor transactions.

6.4.4 SNARK-Themis: A SNARK-based variant

We now detail a more theoretical design, SNARK-Themis, which uses generic arguments of knowledge (specifically, SNARKs [32, 143] for NP languages) in a black-box way to verify computation. Intuitively, for SNARK-Themis, instead of forwarding the replica local orderings to all replicas, the leader will now create a SNARK to prove correctness of its proposal.

Through this, SNARK-Themis achieves optimistic $\mathcal{O}(n)$ communication. Notably, we find that existing fair-ordering protocol designs cannot easily achieve the same even using SNARKS. This makes SNARK-Themis the *first such fair-ordering protocol*; the communication here is asymptotically optimal and

equivalent to state-of-the-art consensus protocols without fair-ordering guarantees.

SNARK preliminaries. For a language L in NP with witness relation $\mathcal{R}_L$, a SNARK for $\mathcal{R}_L$ is a tuple of efficient algorithms $(\mathsf{Gen}, \mathsf{Prove}, \mathsf{Verify})$ where Gen generates the trusted parameters pp (e.g., the CRS) given 1^λ where λ is the security parameter for the argument system, Prove is the prover algorithm, and Verify is the verifier algorithm. Given pp and any $(x, w) \in \mathcal{R}_L$, $\mathsf{Prove}(pp, x, w)$ produces a proof π attesting that $x \in L$. The proof can be checked using $\mathsf{Verify}(pp, x, \pi)$. We require the standard completeness, soundness, and knowledge properties for SNARKs. For (asymptotic) efficiency of our protocol, the SNARKs need to have constant-sized proofs excluding a poly-log factor in the security parameter (e.g., [143, 179]). We do not require any zero-knowledge property.

SNARK proof for the leader proposal. Consider the language L in NP such that (x, w) is in the witness relation R_L if the following holds: x is a string representation of directed graph $\mathcal{G}$, w is parsed into $\mathsf{List}_{i_1} \| \cdots \| \mathsf{List}_{i_{n-f}} \| \sigma_{i_1}(\mathsf{List}_{i_1}) \| \cdots \| \sigma_{i_{n-f}}(\mathsf{List}_{i_{n-f}})$ where each $(\mathsf{List}_j, \sigma_j)$ represents a transaction ordering from a distinct node along with a signature from the node, and $\texttt{FairPropose}(\mathcal{L})$ outputs $\mathcal{G}$ where $\mathcal{L} = \{\mathsf{List}_{i_1}, \ldots, \mathsf{List}_{i_{n-f}}\}$. The string representation can be decided upon deterministically, for example, by choosing vertices in alphabetical order. We use general-purpose SNARKS for the language L to prove correctness. A similar correctness proof can be constructed for the update list $\mathcal{E}_{\text{updates}}$.

Finally, the overall proposal block B by the leader will be the tuple $(\mathcal{G}, \mathcal{E}_{\text{updates}}, \pi = (\pi_1, \pi_2))$ where $\mathcal{G}$ is the graph for newly proposed transactions,

$\mathcal{E}_{\text{updates}}$ is the set of update edges, π_1 is the SNARK for $\mathcal{G}$, and π_2 is the SNARK for $\mathcal{E}_{\text{updates}}$. The upshot is that now, the size of B no longer depends on the number of nodes, enabling the underlying consensus algorithm to achieve communication complexity $\mathcal{O}(n)$.

SNARKs for Wendy [170] and Aequitas [161]. Wendy and Aequitas both require an $\mathcal{O}(n^2)$ communication phase where all nodes gossip transactions to all others. This is critically necessary for censorship resistance, and we found no easy way to improve the communication complexity using SNARKs.

Improved complexity for Pompē [259]. The Pompē protocol has a communication complexity of $\mathcal{O}(n^2)$. The general design of Pompē can be modified to achieve optimistic linear complexity through the use of SNARKs for verifying the computation of the median timestamp. Specifically, after the ordering phase, in Pompē, the median timestamp for a given transaction is computed and transmitted during the consensus phase. This is done by the client but Pompē assumes that the all clients are protocol nodes. In this case, the median computation can be proved (by the client) using constant-size SNARKs which enables linear communication in the optimistic case.

However, in standard deployments, *clients and protocol nodes can be separate entities* and not all clients need to be present during the system initialization. In such a case, SNARKs with trusted setup might not provide an appropriate trust model for clients. Current constant-size SNARKs, however, all require trusted setup. Further, even if a trusted setup can be used, the public parameters (e.g., the CRS) need to be communicated to clients. Since the CRS size is dependent on n for known constant-size SNARKs, this means that in the standard setting

where clients and nodes can be distinct, Pompē will not have optimistic communication linear in n.

While SNARKs without trusted setup can be used instead, current state-of-the-art constructions [228, 245, 50] all have at least logarithmic proof sizes. This means that, even with the use of SNARKs, at best, $\mathcal{O}(n \log n)$ optimistic communication complexity can be achieved.

6.5 Implementation and Benchmarks

We ran an extensive set of experiments for Themis that we detail in this section. In addition to the standard performance benchmarks for throughput and latency (Section 6.5.1), we also design a suite of fairness experiments (Section 7) which are useful in quantifying the extent of fair transaction ordering. We begin with an overview of our implementation below.

Implementation details. Our implementation for Themis is bootstrapped from the Hotstuff protocol [256]: a state-of-the-art leader-based protocol for the partially synchronous setting. For this, we started from the authors' open-source libhotstuff codebase [174] (which implements the $\mathcal{O}(n^2)$ version of the Hotstuff protocol). Our primary code changes were having the leader generate fair transaction sequences and having the replicas validate them. We use this implementation to benchmark the performance of Themis.

Remark 7 (Client communication)**.** Traditional consensus research often ignores client communication: Transactions are typically abstracted to originate from the consensus nodes themselves. For fair-ordering protocols, however, it is

important to model client transaction submission. In Themis (as in other fair-ordering protocols), an honest client submits its transaction to all nodes.

Such models have been considered before. For instance, censorship resistance already requires submission to $f + 1$ nodes (since f corrupt nodes could just ignore the transaction). Furthermore, consistent broadcast protocols (e.g., [26]) also require transaction submission to all nodes (to obtain $n - f$ signatures).

In the context of current blockchain infrastructure, clients often communicate through RPC nodes, which can create a centralization vector. Orthogonal to our context, RPC centralization is a well-known issue and efforts to decentralize the infrastructure have independently been proposed and deployed [213, 34]. We also note that for our purpose, a basic modification to client-side software (e.g., in Metamask) is sufficient to allow a client to send its transactions via multiple RPCs or directly to (permissioned) committee members.

6.5.1 Performance and Benchmarks

Experimental setup. For our performance experiments, we consider two settings: (1) Same Region: All nodes are in the same AWS region (us-east-2); (2) Geo-distributed: Nodes are distributed across 5 regions (us-west-1, us-east-1, ap-northeast-1, ap-northeast-2, eu-central-1) with an equal number of nodes in each region. To demonstrate scalability, we vary the total number of nodes n in the system from 5 to 100. Each node is run on an AWS EC2 C5.4xlarge instance with 16vCPUs and 32GiB memory. In addition, we utilize separate "client" nodes to generate and transmit transactions.

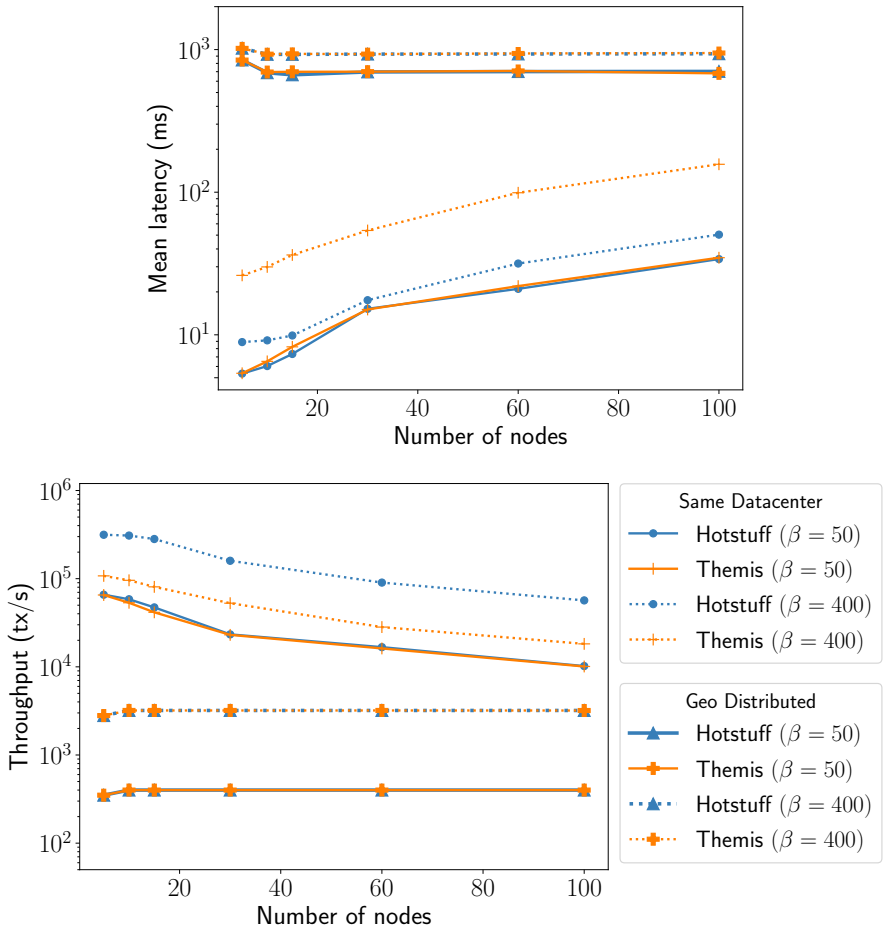

Figure 6.5: Performance comparison of Themis and Hotstuff in both the same-datacenter and geo-distributed settings.

Latency and throughput. We report on the latency and throughput of Themis and compare them to Hotstuff run as a baseline (see Figure 6.5). We experimented with two blocksize (denoting the number of transactions in a proposal) parameters $\beta = 50$ and $\beta = 400$.

Overall, Themis provides very comparable performance to Hotstuff. Notably, we found that both latency and throughput of Themis scale in the same way as Hotstuff as the number of nodes increase. Using a larger blocksize increases the performance gap between the two protocols; this is because Themis needs to build a graph with $\mathcal{O}(\beta^2)$ edges. In the single datacenter setting, this

gap was found for blocksize $\beta = 400$. This difference is not fundamental however, and we highlight that the computation can be parallelized and therefore, by using more cores per node, the performance of Themis can be made comparable to Hotstuff even for a large β. An optimized implementation of the graph algorithms used by Themis could also further boost the performance.

More importantly, we found that this difference vanishes in the geo-distributed setting due to already larger communication latency. In fact, we found that performance of both systems was identical even for very large blocksizes ($\beta = 1200$).

In essence, we expect the performance of Themis to be sufficient for most applications that require its fairness properties.

Performance comparison to other fair-ordering protocols. We do not directly compare the performance of Themis to other fair-ordering protocols; this is primarily due to lack of comparable implementations. For instance, Aequitas and the protocol from [54] have not been implemented, while an open-source implementation of Wendy only provides simulations and is not yet integrated with the consensus layer. Pompē is implemented using Hotstuff as the underlying protocol; their benchmarks also show comparable performance to Hotstuff (similar to Themis), and even better performance in some geo-distributed deployments, due to a fast *ordering* phase that precedes the main consensus layer. However, since this ordering phase itself creates censorship issues within Pompē, a performance comparison would not be on an equal footing.

CHAPTER 7

SUITE OF FAIR ORDERING EXPERIMENTS

To better analyze and compare different fair-ordering definitions and protocols on the same footing, we propose a general suite of *fairness experiments*. Through these, we compare the fairness definitions from [259, 170] as well as receive-order-fairness, and batch-order-fairness. We also compare Themis to other fair-ordering protocols.

Comparison axes. Our analysis comprises primarily of three axes—each targeted at specific setting or attack vector. We start by considering an *ideal* setting where all nodes are honest to understand the best-case scenario (Section 7.3). Our next two experiments simulate adversarial environments. In Section 7.4, we analyze susceptibility to *insertion* (e.g., front-running) attacks. In Section 7.5, we evaluate robustness to *reordering*—to what extent adversarial nodes can influence the final ordering by reordering honest user transactions.

As a related analysis point, in Section 7.6, we also evaluate censorship-resistance from a formal standpoint; here, we show a subtle censorship issue with Pompē [259].

7.1 Fair-Ordering Definitions

Fair ordering definitions from [259, 170]. Zhang et al. [259] and Kursawe [170] consider similar definitions of fairness which we consolidate into a single property—*fair separability*.

Definition 7.1 (Fair Separability)**.** Suppose that tx and tx′ are received by all

protocol nodes. If *all* honest nodes receive tx before *any* honest node receives tx′, then all honest nodes output tx before tx′.

This property is called *timed-relative-fairness* in [170]. It is also stated as a desideratum in [127] although no protocols achieving this property are given. The *ordering-linearizability* notion considered in [259] is similar except that it is only applied when tx and tx′ are both output by the protocol. In particular, ordering-linearizability considers it acceptable if only tx′ is ouput and tx is not, even when the antecedent of the above definition is true, i.e., all honest nodes received tx before any honest node received tx′. By doing so, the Pompē protocol is able to ensure that the delivery of tx′ is not held back by tx even when tx is stuck in a slow network. However, by making this tradeoff, another problem is introduced: a network adversary or even a Byzantine leader node is now able to *censor* a specific transaction from being delivered; this is potentially far more problematic, especially for our primary motivation of DeFi. We discuss this further in Section 7.6.

Comparing batch-order-fairness **on an equal footing.** While batch-order-fairness allows transactions to be output together, this is done only for those in the same cycle, and even within a batch, a total ordering will be enforced later for execution. Therefore, to quantify the strength of batch-order-fairness in the most conservative way, in all our comparisons, we consider the final *total execution ordering* it guarantees, i.e., exactly the fairness property of Themis.

For instance, if $\mathcal{A}$ can adversarially get tx *executed* before tx′ (e.g., through frontrunning or order-manipulation), we count it as a victory for $\mathcal{A}$ and a failure of batch-order-fairness and our protocol. In fact, even if $\mathcal{A}$ is able to place tx and

tx′ into the same cycle when they should not have been, we will still consider it a success for the adversary. We emphasize that even with this conservative approach, our results highlight the strength of batch-order-fairness and Themis in preventing order-manipulation attacks.

7.2 Simulation Environment

We created an environment to simulate the creation and network broadcast of transactions to better understand the effect of different parameters. This is useful for both our ideal setting (Section 7.3) and adversarial reordering (Section 7.5) experiments. Transactions are generated by a *sending process* with the time delays between consecutive transactions sampled from the distribution GenerationDist. For each transaction, we simulate when it would reach different consensus nodes by sampling the network latency from a distribution NetworkDist. Let $\mathsf{Send}(\text{tx})$ denote the time that tx was generated and $\mathsf{Recv}(i, \text{tx})$ denote the time that tx is received by replica i.

We instantiate each distribution as an exponential distribution, which is standard in networking literature. The rate of generation or arrival of messages is modeled as a Poisson process, which is equivalent to the intervals between messages (i.e., GenerationDist) being an exponential distribution. Separately, the network delay is also usually modeled as an exponential distribution. We set GenerationDist to the exponential distribution $\mathsf{Exp}(1/\mu)$ with mean $\mu = 1$, and chose NetworkDist to be *independently distributed* with mean $r\mu = r$ for a *network ratio* parameter r.[1]

[1]The choice of $\mu = 1$ is w.l.g. since exponential distributions satisfy the scaling property — If $X \sim \mathsf{Exp}(1/\mu)$, then $kX \sim \mathsf{Exp}(1/k\mu)$.

The network ratio r represents how quickly new transactions are created compared to their propagation time. It serves as a proxy for how far apart consensus nodes are from one another; a small r ($\ll 1$) captures a setting where all nodes are in the same local network, while a larger r (say 10 or 100) is typically more reflective of a geo-distributed setting.

We highlight that even when the network delay is very large, transaction latencies with smaller r can be approximated by, abstractly, setting a coarser *granularity* for fairness. We include a discussion below. In our experiments, we assume the finest possible granularity since this is the most challenging setting, but we note that in some cases, a coarser granularity may be acceptable.

Remark 8 (Network ratio and fairness granularity)**.** We highlight that even when the network delay is very large, transaction latencies with smaller r can be approximated by, abstractly, setting a coarser granularity for fairness. A granularity of g (defined below) is equivalent to only quantifying fair ordering across these width g intervals, and ignoring unfairness of transactions within the same interval. Note the time intervals can be w.r.t. the local time at a given node, and synchronized clocks are not required. In other words, the granularity can be changed without affecting any protocol assumptions.

Definition 7.2 (Fairness granularity)**.** For granularity g, we consider that timestamps are bucketed into slots of interval g time each (e.g., $[0, g)$, $[g, 2g)$ and so on). Events within a time bucket are assumed to happen at the same time.

Transaction comparisons. It is most fruitful to analyze the ordering for transactions sent around the same time. The scenario where tx′ is sent after tx has already been ordered, for instance, is not particularly interesting. Therefore we

take the approach of studying sets of temporally clustered transactions, rather than workloads of extended duration; we will use sets of 1000 such transactions.

7.3 Ideal Setting Comparison

We first seek to understand the utility of different fair-ordering properties *in an ideal, non-adversarial setting*. Here, the only influence is from how far nodes are and any randomness in network propagation. In this ideal setting, we ask how close we can get to the magical first-in-first-out property of ordering transactions based on when they were *sent*.

Ideal ordering summary. We vary r from 10^{-2} to 10^{3}, and measure the number of transaction pairs that are correctly accounted for by different fairness definitions, i.e., their ordering is consistent with the sending process: tx is ordered before tx′ when $\mathsf{Send}(\text{tx}) < \mathsf{Send}(\text{tx}')$. To quantify the usefulness of batch-order-fairness conservatively, we consider it a failure of the definition if transactions end up in the same cycle. Overall, we still find that both receive-order-fairness and batch-order-fairness ensure closer to ideal ordering than fair separability for all values of r (see Figure 7.1).

We also find that receive-order-fairness and batch-order-fairness perform identically except for when $\gamma \approx 0.5$. The deviation between the two definitions occurs due to (strong)-Condorcet cycles (since we are in an ideal setting); intuitively, this shows that these cycles are infrequent. To underscore this, we further investigate these cycles; we find that cycles are rare, and are of small length even when they do arise (see Figure 7.2). This effectively demonstrates that

even though the stronger receive-order-fairness is impossible without network synchrony, in many practical settings, the performance of batch-order-fairness is almost identical.

Comparing receive and batch order-fairness. For $n = 100$, note the plots for the two definitions are identical except for when $\gamma = 0.51$. Even for $\gamma = 0.51$, they are identical for a small r. When $n = 20$, the two definitions also deviate for $\gamma = 0.6$. This is because Condorcet cycles are much more common when r is large and when γn gets close to $n/2$—for large n, this effect is not seen until γ gets close to 0.5.

Abstractly, this deviation can be seen as the result of the frequency with which Condorcet cycles arise in practice. Note that the presence of cycles also means that receive-order-fairness cannot be achieved in these settings. Therefore, a natural question to ask now is how commonly do cycles arise, and when they do, how large do they get? We found that in practical settings, Condorcet cycles are small, and therefore, effectively the two definitions are quite similar.

Cycle size for order-fairness. We find the number of transactions that are present in (strong)-Condorcet cycles of each size in the ideal setting. Figure 7.2 contains results for $n \in \{20, 100\}$ and for reasonable network ratios $r \in \{1, 10\}$. For example, for a given length-k cycle, we count k transactions in the bucket for k. Note that $k = 1$ means that there is no cycle.

We found that for $r = 1$, there were no cycles found for $\gamma \in \{1, 0.8, 0.6\}$. For $\gamma = 0.51$, for $n = 20$, there was one 3-length cycle and one 4-length cycle, while there was just one 3-length cycle for $n = 100$.

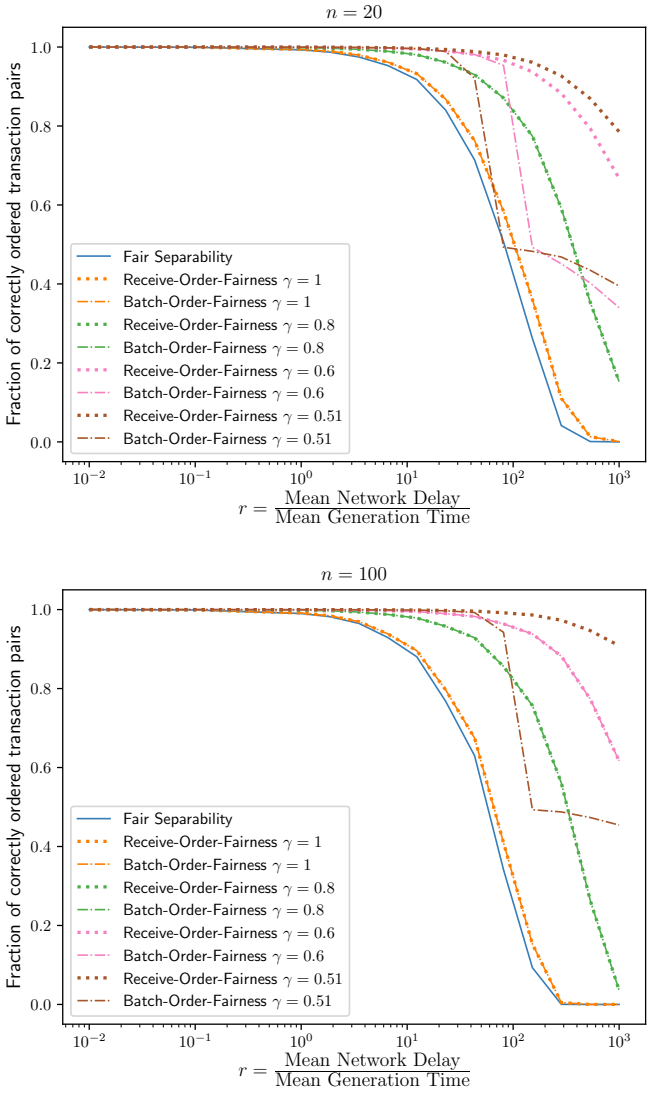

Figure 7.1: Comparison of the fraction of transaction pairs correctly ordered by different fairness notions as a function of r.

When $r = 10$, we did not find cycles for $\gamma \in \{1, 0.8\}$. There were a few length-3 cycles for $\gamma = 0.6$ for $n = 20$ but none for $n = 100$. For $\gamma = 0.51$, cycles are more common but still a reasonable network ratio of $r = 10$, the max cycle length was 11 for $n = 20$ and 5 for $n = 100$. We also notice that it is less common to find cycles (for the same γ) as n increases due to a smaller variance.

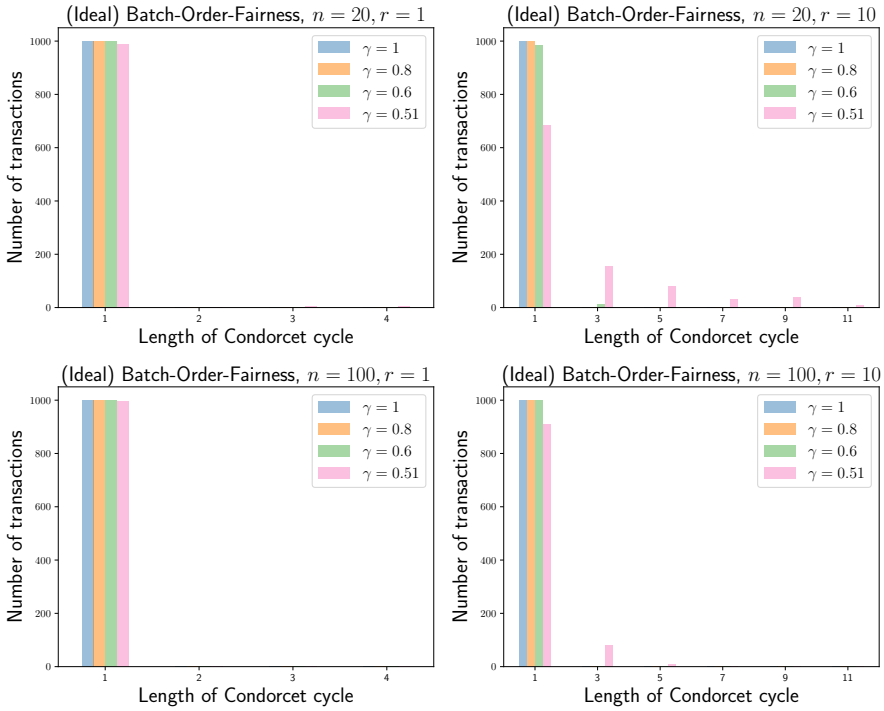

Figure 7.2: Comparison of the number of transactions in a cycle of specific lengths. A value of 1 means that there is no cycle.

7.4 Network Level Insertions and Frontrunning

We now analyze how robust a fairness definition is to network-level *frontrunning*, which is arguably the core ordering issue in today's networks. We define network-level frontrunning as a node Y being able to perform the following attack: On receiving a transaction tx from another node X, it attempts to create a new transaction tx′ and get it sent to other nodes in an attempt to get tx′ ordered before tx. Note that this frontrunning can take place before transactions are received by all nodes, i.e., even before the consensus protocol begins.

While causal ordering or privacy techniques can help hide transaction data before ordering, thus thwarting targeted frontrunning (i.e., based on transaction content), they do not help against problems such as metadata (e.g., IP address)

leakage, or non-targeted frontrunning (i.e., based only on the transaction existence rather than its content). Therefore, quantifying network frontrunning remains important. Notably, our experiment measures frontrunning protection even in cases when transaction privacy is not sufficient. Nevertheless, we note that all fair ordering protocols can incorporate privacy as a complementary protection—e.g., as a backstop against fair-ordering failures in extreme settings such as full adversarial network control. We also emphasize that our experiment can be easily adapted to other network-level attacks like backrunning, sandwiching, and priority attacks (e.g., first in line for an ICO [182]).

7.4.1 Formal analysis in a natural network setting.

As a concrete result, we show how Themis as well as the order-fairness definitions more broadly prevent frontrunning in a natural network setting where triangle inequality is respected. Abstractly, we prove that in this model, an adversary cannot force a frontrunning transaction tx_{adv} into the same Condorcet cycle as an honest transaction tx, and consequently will not be able to get tx_{adv} executed earlier. This serves to complement our experimental analysis of frontrunning in real network settings.

In a network where triangle inequality holds, an adversary cannot see an honest transaction and then create and send an adversarial transaction (i.e., an attempt to frontrun) in a way that any honest node receives the adversarial transaction first. In other words, if tx is an honest transaction and m is an adversarial transaction sent afterwards, then all honest nodes should receive tx before m. In this network setting, we can show that Themis prevents frontrunning—

i.e., the adversarial transaction will be ordered strictly after the honest transaction it is trying to frontrun. We show this through the following lemma.

Lemma 7.3. *An adversary cannot force an adversarial transaction m to be in the same cycle as any honest transaction under the above mentioned network model.*

Proof. Deferred to Appendix D.2. □

Remark 9 (Full adversarial control)**.** We remark that if the adversary $\mathcal{A}$ has full network control of when transactions (both honest and adversarial) are *input* to honest nodes, then network-level frontrunning is trivial in any protocol. Note that even privacy techniques do not fully prevent this—$\mathcal{A}$ can still perform non-targeted and metadata-based frontrunning. Therefore we explore real-world networks in our analysis instead of ones with full adversarial control.

More specifically, the usual consistency and liveness properties remain intact even if $\mathcal{A}$ has such a full network control. Interestingly, even the fair-ordering property holds based on the input orderings, but since $\mathcal{A}$ can essentially manipulate the input orderings of honest nodes themselves given full network control, the protocol property no longer reflects any intuitive notion of fairness.

Indeed, this limitation led to Kelkar et al. [161] formalizing two networks: the (standard) *internal* network (for communication amongst consensus nodes) and the *external* network (for all transaction submission). The adversary has full control over the internal network but does not control the external network. [161] notes that such a power would be similar in spirit to controlling e.g., the user's access to the internet. This ensures that the input orderings of honest nodes cannot be manipulated by the adversary. We highlight that similar assumptions of the adversary not having full control over transaction submission

are present in all previous works [157, 259, 170].

7.4.2 Experimental analysis.

Setup. We used a real network rather than a simulation for this experiment to understand the possibility of frontrunning in practice. Our experiment studies frontrunning at the network layer; we do this to separate out the impact of a network adversary from adversarial transaction reordering through controlling consensus nodes (studied in Section 7.5). Through measurement of communication latency between nodes, our goal now is to find the number of *frontrunnable pairs*—i.e., (A, B) such that B is able to frontrun transactions originating from A. While we experimentally find the number of such pairs, we emphasize that in practice, the success of frontrunning also depends on other factors—for instance, the specific protocol, the expected profit of a particular frontrunning strategy, the variance in network latency, the presence of competing entities etc.

Frontrunnable pairs. We now describe frontrunnable pairs for the fairness notions considered. For fair separability, frontrunning is possible if there exists nodes A, B, C, D (where A, B, and C are distinct, D is different from A and B but potentially the same as C) such that $\mathsf{ping}(A, B) + \mathsf{ping}(B, C) < \mathsf{ping}(A, D)$. When this happens, notice that (A, B) is frontrunnable since B can receive a transaction tx from A (working as a client) and forward its own adversarial transaction tx′ to C before tx was received by all honest nodes. Here, fair separability will not apply since the antecedent is violated—there is an honest node (C) that received tx′ before some other honest node (D) received tx.

For (receive/batch)-order-fairness, (A, B) is frontrunnable if there are at least $n(1-\gamma)+f+1$ nodes C such that $\mathsf{ping}(A,B)+\mathsf{ping}(B,C) < \mathsf{ping}(A,C)$. This is because tx sent by A is no longer receiver before tx′ by at least $\gamma n - f$ nodes. Frontrunning success is guaranteed if there are more than $\gamma n - f$ such nodes C. Observe that each satisfying (A, B, C) corresponds to breaking triangle inequality. Therefore, the possibility of frontrunning here for (A, B) is synonymous to triangle inequality being violated in the network for a *non-trivial* (at least $f+1$) number of nodes. We find that is quite unlikely to hold for real-world networks. For order-fairness, we report on both the $\gamma = 1$ case (for which triangle inequality needs to be violated $f+1$ times for the pair to be frontrunnable), and the optimal case (for which it needs to be violated $> n/2$ times).

Experimental results. We started with a simple toy setup by deploying one node each in five different AWS regions. We defer the details to Appendix D.2 and directly proceed here with our analysis using a large scale network with 250 servers.

For this, we used the publicly available WonderProxy dataset [251] which contains ping times (in both directions) between each server pair averaged over 30 pings per hour over the two day period of July 19-20 2020. Now, for both $n = 20$ and $n = 100$, we randomly sample an n-sized committee from the 250 servers and compute the number of frontrunnable pairs for different fairness definitions. We average the results over 100 randomly chosen committees. Table 7.1 reports on our findings.

For fair separability, on average 82% ($n = 20$) and 94% ($n = 100$) pairs were found frontrunable. In contrast, for order-fairness, the corresponding numbers

Setting	Total Pairs	Number of frontrunnable pairs		
		Fair Separability	Receive/Batch Order-Fairness ($\gamma = 1$)	Receive/Batch Order-Fairness (Optimal)
5 AWS nodes	20	10	0	0
Random $n = 20$	380	313	14	1
Random $n = 100$	9900	9326	262	16

Table 7.1: Number of frontrunnable pairs for fair separability and order-fairness for several settings. The first line denotes to our example setting with 5 geo-distributed AWS nodes. For the other rows, we randomly sample n-sized committees from the Wonderproxy dataset [251]. We chose $n = 20, f = 4$ and $n = 100, f = 24$ for the second and third row respectively.

were 4.5% and 2.8% for $\gamma = 1$ and 0.26% and 0.16% in the optimal case. Notably, frontrunning gets easier on larger networks for fair separability while it gets harder for order-fairness.

7.5 Robustness to Adversarial Reordering

In this section, we evaluate to what extent adversarial nodes can influence the final transaction ordering through *reordering*, i.e., by claiming within the protocol to have received user transactions in a different order than they were actually received.

Experiment intuition. All fair-ordering properties guarantee some robustness to adversarial reordering. For instance, when all honest nodes receive tx before any honest node receives tx′, fair separability guarantees that tx will be output before tx′ no matter what the adversary does. Yet this says nothing about *how often* the fairness property will apply for transaction pairs or what happens when the property *does not apply* (e.g. is the fairness all or nothing or does it smoothly

degrade?). Understanding this will paint a more complete picture of the overall robustness of a protocol to adversarial reordering.

Dependence on transaction *closeness*. Zhang et al. [259] show that it is impossible to make the final ordering completely independent of the adversary (since intuitively, adversarial nodes are indistinguishable from honest nodes). Still, ideally, a good ordering protocol restricts adversarial influence only to transactions that are already "very close" in the honest ordering — close in the sense that even their honest ordering could have been inverted by small network delays. Effectively, this would signify that the impact of the adversary is similar to that of small network fluctuations.

As a proxy for "closeness," we use the distance metric

$$\mathsf{Dist}(\mathsf{tx}, \mathsf{tx}') = |\#_{(\mathsf{tx}<\mathsf{tx}')} - \#_{(\mathsf{tx}'<\mathsf{tx})}|$$

where $\#_{(x<y)}$ denotes the number of nodes that received x before y. Dist varies from $(n \bmod 2)$ to n in increments of 2. A small Dist implies a "fragile" pair, where the number of nodes that received tx earlier is roughly the same as the number that received tx′ earlier, while a large Dist means that one of the transactions was received earlier by a large number of nodes.

Adversarial strategy. Since the space of possible adversarial strategies is practically unbounded, we consider just one specific yet powerful attack in which the adversary will attempt to flip the ordering of transactions. This adversarial strategy is quite simple: an adversarial node simply flips the ordering of all transactions it received as input. For instance, if it received transactions in the order $[\mathsf{tx}_1, \ldots, \mathsf{tx}_l]$, it claims to have received them in the order $[\mathsf{tx}_l, \ldots, \mathsf{tx}_1]$.

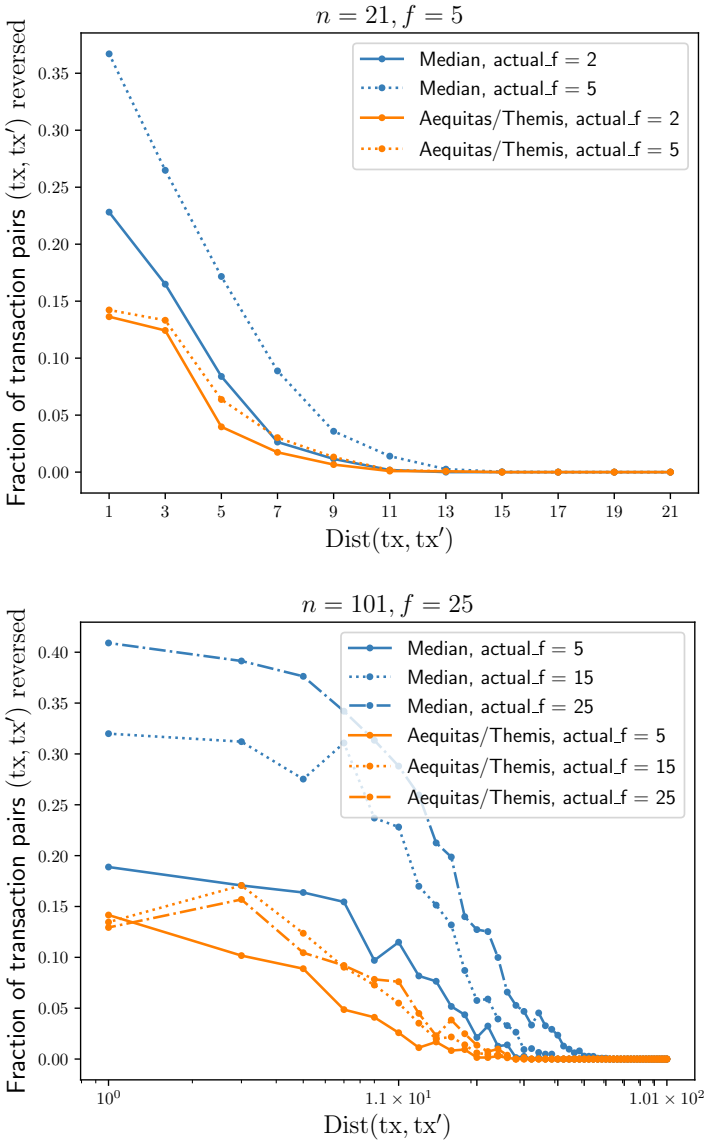

Figure 7.3: Adversarial influence on transaction ordering

Despite the simplicity, this should be a reasonable strategy for attempting to reverse the final ordering for many transaction pairs. It generalizes attempting to flip the ordering of a single transaction pair. Still, we acknowledge that better adversarial strategies likely exist and fair ordering protocols should also be tested against them. We leave this for future work.

Measurement. Given our adversarial strategy of transaction flipping, we now find how frequently it enables the adversary to reverse the ordering (w.r.t. the honest ordering) of a transaction pair with a given distance. As a more accurate comparison, for Themis, even if the adversary is able to put transactions into the same cycle when they should not have been, we consider as a failure for our protocol.

We consider two settings: $n = 21, f = 5$ and $n = 101, f = 25$, and vary the *actual* number of corruptions. In each instance, we find the fraction of transaction orderings the adversary successfully inverted. Figure 7.3 shows the results. We compare Themis ($\gamma = 1$) to a simple *median* protocol which sorts transactions by their median timestamp. Note that this protocol abstracts exactly the relevant fairness component of Pompē [259] and Wendy [170].

Reordering insights. We first notice that for large $\mathsf{Dist}(\mathsf{tx}, \mathsf{tx}')$ (≥ 13 for $n = 21$, and ≥ 71 for $n = 101$), there was no successful reordering against either protocol, even with the maximum number of corruptions (for Themis, this was true for all $\mathsf{Dist} \geq 11$ for $n = 21$ and ≥ 29 for $n = 101$). Note that the X-axis in the $n = 101$ figure is in log-scale to better highlight reordering for transactions pairs with small distance (since reordering was absent for large Dist).

Also note that in all cases, the fraction of reversed transaction pairs drops sharply as Dist gets larger, which shows both fair ordering protocols working as expected. For small Dist, the transactions are so close that even non-adversarial random network delays can reverse their ordering, and thus it is unreasonable to expect resilience to adversarial strategies.

Still, overall, we found that for a given number of actual corruptions, it is

much easier to reverse the ordering for a median timestamp protocol than for Themis. In fact, in our experiment, it was easier to reverse the ordering against the median timestamp protocol even through a small number of corruptions than it was against Themis using the maximum number of corruptions. Further, we also note that for this specific adversarial strategy, the additional gain against Themis through extra corruptions is small.

7.6 Censorship Resistance and Attacks

Censorship resistance is defined as the property that if an honest client sends a transaction to the consensus nodes, then it will eventually be output by the protocol. While many existing protocols provide censorship resistance (e.g., [192]), we emphasize that on its own this does not guarantee that the transaction *ordering* is "fair."

We find that some fair-ordering designs can have inherent tradeoffs between liveness and censorship resistance; this is seen when an earlier phase is used to first *order* transactions prior to consensus.

Censorship attack on [259]. Pompē [259] uses a pre-protocol (specifically, an ordering phase prior to consensus) to compute the median timestamp for transactions, with the fair ordering being taken according to the ascending order of the median timestamp. For $n \geq 3f+1$, as long as $(\mathsf{tx}, \mathsf{tx}')$ satisfies the antecedent of fair separability (Definition 7.1), the median of tx will be smaller than that of tx′. In other words, as long as both transactions *complete the ordering phase*, tx will be ordered earlier by the protocol; this condition is explicitly stated in the

ordering linearizability definition used in Pompē. Unfortunately, this results in censorship resistance being guaranteed only for transactions that pass the pre-protocol phase.

For Pompē, in a partially synchronous network, an adversary can delay sending the median timestamp for tx long enough until tx′ is delivered. If this happens, since the rest of the output is fair, tx can never be output by the protocol. Concretely, Pompē assumes a known upper bound Δ_2 on the time for an honest node to complete the ordering phase; Δ_2 is the maximum time a transaction in the consensus phase waits for any potential transaction from earlier to finish the ordering phase. When this bound is not satisfied, while safety is not broken, earlier transactions are now essentially censored since the consensus phase has moved on to a later timestamp even though they were timestamped during the ordering phase. In order words, to achieve censorship resistance, Pompē needs to assume a known synchronous bound *throughout* the protocol on the time for an honest node to complete the ordering phase, which cannot be assumed in the partially synchronous setting. The synchrony requirement is also especially of concern in the common setting where clients (rather than nodes) submit transactions since the bound would now have to involve the client's connection.

While this was partially acknowledged in [259] as a possible way a Byzantine node could selectively chose which of its transactions to submit to the consensus phase, the stronger censorship attack vector was not identified. In the context of DeFi, both selective disclosure and censorship are far more severe. This also surfaces an inherent tradeoff between liveness and censorship resistance within such protocols. Since the client (or the protocol node in charge of the transac-

tion) is in charge of accumulating timestamps from the nodes in the ordering phase before proceeding to the consensus phase, as [259] notes, it is not possible to determine whether the transaction was sent to the consensus phase in time or not. In fact, as a result, a view change also cannot be triggered in order to restore censorship resistance. Thus either there will be a liveness failure (by potentially waiting forever), or a lack of censorship resistance.

From our analysis, the core reason for this is the separation between the pre-protocol that computes the median timestamp and the consensus phase, along with the design that a single node is responsible for moving a given transaction from the former to the latter phase. Here, due to the presence of Byzantine nodes, it is impossible to ensure that any transaction that has been timestamped at the protocol nodes also makes its way into the consensus phase.

In contrast, Themis, along with Wendy and Aequitas, are not susceptible to such censorship; if tx is received by all honest nodes, it will not be censored, and moreover will be sequenced fairly.

BEYOND ORDER-FAIR CONSENSUS

The past several chapters introduced order-fair consensus—an extension to the standard consensus abstraction that also provides more robust guarantees on how transactions are ordered. In the past few years, a substantial body of work has proposed new order-fairness definitions, protocols, and optimizations.

The following proposes a few promising extensions, which the community been exploring recently:

1. **(Removing honesty assumptions).** One concern with existing order-fair consensus protocols is that they retain the typical honest (super)-majority assumptions made in the classical distributed systems literature. That is, the protocols maintain their properties only as long as e.g., more that $1/2$ (or $2/3$ or $3/4$) fraction of the nodes remain honest. For blockchain-based applications however, relying on such honesty assumptions may not be appropriate. Indeed, given that order-manipulation in decentralized finance has profit-based motivations, protocol ordering guarantees should be designed to hold even under rational models.
2. **(Verifiable fairness properties).** Existing order-fairness guarantees are not publicly verifiable—i.e., there is no signal given when they fail to hold. In turn, an observer cannot detect when the honest-majority assumption is violated and fairness fails to hold. It is valuable to develop ordering policies that are verifiable as well as accountability techniques for transaction ordering in general.
3. **(Fairness for different market participants).** Order-fairness focuses on ordering guarantees for users. First-in first-out (FiFo) is a natural policy,

which many of the order-fairness definitions from earlier chapters aim to extend to a decentralized setting. But in the context of financial applications, FiFo also creates some negative externalities, as evident by "latency racing wars" in traditional finance. The subsequent Chapter 8 introduces a new ordering policy that retains the useful qualities of FiFo for users, and also minimizes negative externalities caused by arbitrageur incentives.

CHAPTER 8

TIMEBOOST: LATENCY RACING VS BIDDING FOR TRANSACTION ORDERING

This chapter introduces the design of TimeBoost—a practical transaction ordering policy for rollup sequencers that takes into account both transaction timestamps and bids; it works by creating a score from timestamps and bids, and orders transactions based on this score.

TimeBoost is transaction-data-independent (i.e., can work with encrypted transactions) and supports low transaction finalization times similar to a first-come first-serve (FCFS or pure-latency) ordering policy. At the same time, it avoids the inefficient latency competition created by an FCFS policy. It further satisfies useful economic properties of first-price auctions that come with a pure-bidding policy. We show through rigorous economic analyses how TimeBoost allows players to compete on arbitrage opportunities in a way that results in better guarantees compared to both pure-latency and pure-bidding approaches.

The content in this chapter was first published at AFT 2023 [180].

8.1 Introduction

We focus on ordering policies for *centralized sequencers*—meaning that a single *sequencer* receives transactions from users and publishes an ordered sequence to be used for execution. A transaction ordering policy here specifies how the resulting output sequence depends on the contents and arrival times of transactions at the sequencer. Our work provides rigorous economic analyses to justify the utility of our proposed policy.

Why consider a centralized sequencer? In addition to the centralized sequencer setting being a potentially simpler model to study as a first step, there are two other main reasons why we choose to do so in this work:

1. *Existing use-cases are already centralized.* Decentralized blockchains such as Ethereum are still *ephemerally centralized* with respect to ordering—for a given block, similar to a centralized sequencer, only a single miner/validator is in complete control of the inclusion and ordering of transactions within the block. Similarly, current layer-2 "rollup" protocols (such as Arbitrum and Optimism) also employ a centralized sequencer to order transactions in a batch posted to the underlying Ethereum base-chain.
2. *Ordering policies are mostly orthogonal to the problem of sequencer decentralization*.

   While decentralizing the sequencer is an important active research direction, we note that a suitable transaction ordering policy can be chosen orthogonally to the method of sequencer decentralization. In particular, the decentralized protocol can first be used to agree on single *pre-ordering* or *scoring* of transactions, following which a specific ordering policy can be applied. In other words, the output of the decentralized protocol can be thought of simulating the input of a virtual centralized sequencer on which the ordering policy gets applied.

   An example of this is seen in the recent line of works on fair-ordering [161, 158, 54, 259, 170]—they can be thought of as a decentralized implementation of a first-come-first-serve ordering policy which combines local transaction orderings from many nodes.

Furthermore, while current centralized sequencer implementations are

semi-trusted in that they receive transactions in plaintext and are expected not to deviate from the specified ordering policy or insert transactions of their own, we note that transaction data can be hidden from the sequencer by using threshold decryption by a committee (i.e., the sequencer only sees encrypted transactions and orders them, only after which a committee decrypts the plaintext) or trusted hardware (such as Intel SGX). Through these techniques, the adversarial behavior of the sequencer can be substantially restricted.

The study of ordering policies is important even when the sequencer is trusted (or is suitably constrained as mentioned above) due to the presence of other profit-seeking entities in the system. For instance, after the sequencer publishes state after execution of previous transaction(s), arbitrage opportunities can be created; players in the system will compete with each other to take advantage of these opportunities. Similar situations can also arise due to state updates from external systems.

8.1.1 Existing Ordering Policies

Ordering policies used on blockchains today fall roughly into three categories described below.

First-come first-serve (FCFS). One natural ordering policy is the first-come, first-serve (FCFS) rule. Here, transactions are sequenced in the same order that they were received from users. There are several advantages to FCFS: to begin, it is simple to implement and seems intuitively fair—after all, it is a commonly used policy even for real-world interactions. FCFS also minimizes transaction

latency: transactions can be continuously sequenced as they arrive, and do not need to conform to the discrete granularity of blocks. The sequencer in the layer-2 rollup Arbitrum employs an FCFS policy.

One major disadvantage of FCFS however, is that creates *latency competition* in the sense that entities are incentivized to position themselves as close to the sequencer as possible in order to be the first to react to any new market information. This is a well known and studied problem within traditional financial systems. Indeed, high frequency trading (HFT) firms invest millions of dollars into low-latency infrastructure that can operate sub-microsecond or even finer scales; their trading accounts for roughly half of all trading volume [193]. This inclination to latency investment is highly inefficient since the investment happens externally to the system (as opposed to bidding; see below) and therefore cannot be used beneficially within the system. Recent works [236, 20] have also shown the potential for similar strategic manipulation within a pure FCFS protocol in the decentralized setting.

One crucial point to emphasize here is that this latency competition in FCFS *does not disappear even if transaction data is hidden* (e.g., transactions are encrypted). This is because any state changes (from the sequencer or even from external systems) can trigger a profit opportunity wherein it is beneficial to have the quickest access to the sequencer. As a specific example, an update on the trading price of a token can create an arbitrage opportunity whose profit will go only to the player who can submit its transaction to the sequencer first[1]. This kind of latency-based arbitrage has already been seen in Arbitrum, which implements a centralized FCFS sequencer.

[1] Another approach if the sequencer broadcasts state information in a random order to clients is to create many dummy client copies, thereby increasing the chances that some copy gets the feed faster.

Per-block transaction bidding. A second natural policy is to group transactions into blocks, then order transactions within a block based on their *bid*. Specifically, each transaction is submitted along with a fee or bid; the sequencer now collects all transactions submitted within some time interval and sequences them by the descending order of their bids. This essentially simulates a first-price all-pay auction [169] (i.e., players bid independently; the highest bid wins but all players need to pay their bid amount) to take advantage of a particular arbitrage opportunity. Since players submit their bids independently, the bidding policy can work as expected even when transactions are encrypted (since state or market updates create arbitrage opportunities).

One advantage of a bidding policy (compared to FCFS) is that the payment is internal to the system and therefore can be utilized within it to e.g., subsidize protocol operation costs.

When the block-time is large (e.g., $12s$ as in Ethereum), it is expected that for almost all arbitrage opportunities, all interested players can post their bid within the time interval in an attempt to take advantage of the opportunity. However, when the block-time is small (this is typically the case in layer-2 protocols to increase scalability), perhaps surprisingly, having a connection with lower latency can provide a substantial advantage. This is because when the market update happens close to end of the block time, only players with a faster connection will be able get their transaction included in the block; consequently, they may be able to take advantage of the arbitrage opportunity with a smaller (or even a zero) bid.

Looking ahead, our TimeBoost policy (which combines both arrival times and bidding) will enable arbitrageurs to prefer bidding even when block times

are small, thereby allowing the protocol to capture this value rather than it being lost to external latency infrastructure.

Block or MEV auctions. A third widely-used policy auctions off the complete rights to choose and order transactions within a block. Here, the sequencer does not order transactions itself but rather accepts block proposals from external players (often called block *builders*) and chooses the proposal from the builder who pays the most. These auctions initially arose from the realization that significant profit (often referred to as maximal (previously miner) extractable value or MEV [92, 21]) can be extracted by manipulating the ordering of user transactions. In the past two years, through companies such as Flashbots and Bloxroute, an MEV marketplace has been created on Ethereum *outside of the protocol* to connect block proposers (entities in charge of proposing or sequencing a block) to block builders (players who find MEV opportunities and order user transactions to take advantage of them)—the result has been the extraction of hundreds of millions of dollars in profit from user transactions [216, 246].

While some MEV (such as arbitrage, which provides incentives for price discovery) is benign and can be done without the knowledge of user transactions, other forms of MEV extraction crucially rely on the transaction data. Recent works [246, 254] have shown such MEV to be significantly detrimental to users. The emergence of such MEV extraction has largely been attributed to the rationality of block proposers as well as the lack of regulation. For example, in traditional financial systems, it is often illegal or at the very least heavily constrained to profit from the knowledge of user transactions (for instance, payment-for-order-flow (PFOF): the selling of user transaction data is illegal in the UK, and, while legal in the US, still requires users to be provided with guarantees of "best

execution").

A design goal for our work is therefore to design ordering policies that are data-independent, i.e., they do not use transaction data for ordering. This will allow them to be used even when transactions are encrypted at the time of sequencing.

8.1.2 Our contributions

TimeBoost: **An ordering policy that combines FCFS and bidding.** We propose TimeBoost, an ordering policy that combines both FCFS-style timestamps and first-price auction style bids. Below, we describe several natural goals that went into our design.

1. **Data independence.** The policy should not utilize the transaction data for ordering. This is a natural goal in order to support encrypted transactions and prevent data-dependent MEV attacks on transaction ordering.
2. **Low finalization time.** The policy should be able to sequence transactions within a short time g (the specific parameter can be set according to the application). This is important to improve the user experience with the system since transactions will be sequenced within time g after they are received.
3. **Independence of irrelevant transactions.** The ordering between two given transactions should not depend on the presence of other transactions. This is useful to prevent an adversary from inserting irrelevant transactions that results in flipping the ordering between two target trans-

actions. Importantly, this property also ensures that a transaction submitter's strategy need only consider transactions that are relevant to the party's goals—for example if Alice is trying to capture a particular arbitrage opportunity, she need only worry about other transactions affecting that opportunity.

4. **Inclination to spending via bids instead of latency infrastructure.** As mentioned before, investments into latency infrastructure are highly inefficient from the system standpoint since the value spent cannot be utilized effectively by the system. Therefore, a natural goal is to disincentivize latency investment and instead incentivize players to bid for their transactions. Looking ahead, perhaps surprisingly, we find that the pure bidding policy results in a larger latency competition than our TimeBoost policy which combines bidding with FCFS style timestamps.

TimeBoost **details.** Intuitively, TimeBoost works by assigning *scores* to transactions based on both their arrival times and their bid. The final ordering is taken to be descending in the transaction scores. More specifically, for a transaction with arrival time t and bid b, TimeBoost assigns it the score $S(\text{tx}) = \pi(b) - t$ where intuitively π represents a function for "buying time"—by increasing the transaction bid, users can reduce their effective timestamp (or equivalently, increase their score). Section 8.3 describes how to choose the function π.

Importantly, there is a limit to how much time can be "bought" through the bid—in particular, no transaction can outbid a transaction received some g time earlier. Such a property is required to ensure the quick finalization of user transactions. At the same time, transactions received less than g time before can always be outbid; this means that arbitrageurs always have g time to com-

pete for any arbitrage opportunity as opposed to a pure bidding policy and will therefore prefer bidding over latency infrastructure investments.

We also show that TimeBoost satisfies all the useful economic properties of first-price all-pay auctions. Further, we show that players spend exactly the same amount in total with TimeBoost, as they would spend if only latency investment was allowed, except that most of the investment is done through bidding and therefore can be captured within the protocol for e.g., lowering user fees or for protocol development.

8.2 Ordering Policies

8.2.1 Preliminaries

A transaction tx that arrives at the sequencer can be characterized by a tuple (data, t, b) where data represents the transaction data, t denotes the arrival time, and b denotes the transaction bid (note that when transactions are of different sizes, b can be instead be considered to be a bid per unit size). Let $\mathcal{T}$ denote the set of all possible transactions; in principle this can be infinite or even uncountable (e.g., if arrival times are in $\mathbb{R}^+$) and our results do hold for these cases. For practical use-cases, typically, arrival times can be assumed to be in $\mathbb{Q}^+$ and bids can be assumed to be in $\mathbb{N}^{\geq 0}$.

An ordering policy now defines how a sequencer orders a finite set $\mathcal{T}'$ of transactions that it has received. A formal definition is given below:

Definition 8.1 ((Data-Independent) Ordering Policy)**.** An *ordering policy* (or al-

gorithm) $\mathbb{P}$ takes as input a finite subset $\mathcal{T}' \subseteq \mathcal{T}$ of transactions and outputs a linear ordering $\mathbb{P}(\mathcal{T}')$. For $\mathsf{tx} \in \mathcal{T}'$, let $\mathbb{P}(\mathcal{T}', \mathsf{tx})$ denote the position of transaction tx in the ordering $\mathbb{P}(\mathcal{T}')$. In other words, given $\mathcal{T}'$ and $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}'$, $\mathbb{P}$ outputs tx_a before tx_b if $\mathbb{P}(\mathcal{T}', \mathsf{tx}_a) < \mathbb{P}(\mathcal{T}', \mathsf{tx}_b)$.

A policy is further called data-independent if it does not make use of the transaction data (i.e., it only uses the arrival time and the bid).

Since we want our ordering policies to not be based on the transaction content, we only consider data-independent policies for the rest of the paper. For simplicity, we can therefore represent a transaction tx simply by the tuple $(\mathsf{tx}.t, \mathsf{tx}.b)$. Furthermore, since ties can be broken by some chosen technique, without loss of generality, we can also assume $(\mathsf{tx}.t, \mathsf{tx}.b)$ tuples are unique. While the tie-breaking can be dependent on e.g., transaction ciphertext or meta-data, this does not affect our analysis and therefore can be safely ignored for the purpose of our paper.

8.2.2 Independence of Irrelevant Transactions (IIT)

A useful property for our ordering policy to have is to prevent the ordering decision between transactions tx_a and tx_b to change depending on what other transactions are being ordered; in other words, the ordering decision should not depend on irrelevant transactions. Intuitively, this is done to ensure that an adversary cannot create dummy transactions in order to flip the ordering decision between two transactions, and so that a party's bidding strategy can ignore transactions irrelevant to that party. We define this property of independence of irrelevant transactions (IIT) below.

Definition 8.2 (Independence of Irrelevant Transactions)**.** We say that a policy $\mathbb{P}$ satisfies *independence of irrelevant transactions* (IIT) if for any pair of transactions $\mathsf{tx}_a, \mathsf{tx}_b$ and any pair of finite subsets $\mathcal{T}_1, \mathcal{T}_2 \subset \mathcal{T}$, the following holds:

$$\begin{aligned} &\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_1, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_1, \mathsf{tx}_b) \\ \Leftrightarrow\ &\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_2, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\} \cup \mathcal{T}_2, \mathsf{tx}_b). \end{aligned}$$

8.2.3 IIT Implies a Score-Based Policy

We now show that the IIT property implies that a score-based policy needs to be used—that is, also needs to be independent of the set $\mathcal{T}'$ being ordered.

Intuitively, a score-based policy works as follows: for transaction tx, it assigns a score $S(\mathsf{tx})$ based only on the arrival time $\mathsf{tx}.t$ and the bid $\mathsf{tx}.b$. Here too, scoring ties can be broken in a pre-specified manner. The output sequence is then taken to the descending order of transaction scores. Score-based policies are formally defined below:

Definition 8.3 (Score-based policy)**.** A score is a function $S : \mathcal{T} \to \mathbb{R}$ that assigns to each possible transaction $\mathsf{tx} \in \mathcal{T}$ a score $S(\mathsf{tx})$. An ordering policy $\mathbb{P}$ is called score-based if there exists a score function S such that $\mathbb{P}$ sorts transactions according to S. In other words, there exists S such that for any $\mathcal{T}' \subseteq \mathcal{T}$ and $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}'$, it holds that $\mathbb{P}(\mathcal{T}', \mathsf{tx}_a) < \mathbb{P}(\mathcal{T}', \mathsf{tx}_b)$ if and only if $S(\mathsf{tx}_a) > S(\mathsf{tx}_b)$.

For finite $\mathcal{T}$, we can directly show that IIT implies score-based policies. To show the result for infinite sets, we need to employ the following set-theoretic axiom (defined below) by Cantor [62]. Similar definitions have also been used in in the context of utility theory [98]

Property 8.4 (Cantor's Axiom [62])**.** We say that a pair $(\mathbb{P}, \mathcal{T})$ satisfies Cantor's axiom if there exists a countable set $\mathcal{T}' \subseteq \mathcal{T}$ such that for any pair of transactions $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$ there exists an instance of $\mathbb{P}$ in which some transaction in $\mathcal{T}'$ is ordered between tx_a and tx_b.

Formally there is a finite set $\mathcal{T}'' \subset \mathcal{T}$ with $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}''$ and a $\mathsf{tx}_c \in \mathcal{T}' \cap \mathcal{T}''$ (possibly $\mathsf{tx}_c = \mathsf{tx}_a$ or $\mathsf{tx}_c = \mathsf{tx}_b$) such that

$$\mathbb{P}(\mathcal{T}'', \mathsf{tx}_a) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_c) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_b),$$

or

$$\mathbb{P}(\mathcal{T}'', \mathsf{tx}_b) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_c) \leq \mathbb{P}(\mathcal{T}'', \mathsf{tx}_a).$$

We can now establish the following correspondence between IIT and score-based policies.

Theorem 8.5 (IIT $\Leftrightarrow$ Score-Based)**.** *Let $\mathcal{T}$ denote the set of all transactions. The following hold for any ordering policy $\mathbb{P}$:*

1. *If $\mathcal{T}$ is countable, then $\mathbb{P}$ satisfies IIT if and only if it is score-based.*
2. *If $\mathcal{T}$ is uncountable and $(\mathbb{P}, \mathcal{T})$ satisfies Cantor's axiom, then $\mathbb{P}$ satisfies IIT if and only if it is score-based.*

Proof. It is straightforward to see that a score-based algorithm satisfies the independence of irrelevant transactions (since the score of a transaction depends only on itself and not other transactions).

For the opposite direction, we first prove the second part of the theorem (the uncountable case). We define an order $\prec$ over $\mathcal{T}$ where

$$\mathsf{tx}_a \prec \mathsf{tx}_b :\Leftrightarrow \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_b).$$

Since $\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\})$ is a well-defined for any two transactions $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$, the order $\prec$ is complete and anti-symmetric. By independence and since $\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\})$ is a well-defined order for any three transactions $\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c \in \mathcal{T}$ we have

$$\begin{aligned}
&\mathsf{tx}_a \prec \mathsf{tx}_b \prec \mathsf{tx}_c \\
\Rightarrow &(\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b\}, \mathsf{tx}_b) \text{ and } \mathbb{P}(\{\mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_b) < \mathbb{P}(\{\mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_c)) \\
\Rightarrow &\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_b) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_b, \mathsf{tx}_c\}, \mathsf{tx}_c) \\
\Rightarrow &\mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_c\}, \mathsf{tx}_a) < \mathbb{P}(\{\mathsf{tx}_a, \mathsf{tx}_c\}, \mathsf{tx}_c) \\
\Rightarrow &\mathsf{tx}_a \prec \mathsf{tx}_c
\end{aligned}$$

Therefore, $\prec$ is transitive. We let $\mathsf{tx}_a \preceq \mathsf{tx}_b$ iff $\mathsf{tx}_a \prec \mathsf{tx}_b$ or $\mathsf{tx}_a = \mathsf{tx}_b$.

The Cantor axiom and independence imply that there is a countable $\mathcal{T}' \subset \mathcal{T}$ so that the order $\prec$ satisfies that for any $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$ there is a $\mathsf{tx}_c \in \mathcal{T}'$ such that

$$\mathsf{tx}_a \prec \mathsf{tx}_b \Rightarrow \mathsf{tx}_a \preceq \mathsf{tx}_c \preceq \mathsf{tx}_b$$

By Theorem 1.1 in [66], this, in turn, implies that there is a numerical representation of the order $\prec$ which is a score $S : \mathcal{T} \to \mathbb{R}$ such that for any two transactions $\mathsf{tx}_a, \mathsf{tx}_b \in \mathcal{T}$ we have $\mathsf{tx}_a \prec \mathsf{tx}_b$ if and only if $S(\mathsf{tx}_a) > S(\mathsf{tx}_b)$.

For the first part of the theorem, note that the previous argument also works for a countable $\mathcal{T}$ and in that case we can choose $\mathcal{T}' = \mathcal{T}$ where the Cantor axiom is now trivially satisfied.

□

Remark 10. The above result extends to the case where the policy creates a weak ordering (which can be made strict through a tie-breaking procedure) rather than a strict ordering of transactions. In that case, Definitions 8.2 and 8.3 are adapted to weak orders, and we get a score that might assign the same value to two different transactions. The relaxation to weak orders is useful for the case that the set of transactions is uncountable and not a subset of the real numbers (e.g. if $\mathcal{T} = \mathbb{R}_+^2$). In that case, the Cantor axiom is impossible to satisfy for strict orders but satisfiable for weak orders.

Discussion. We note that in our context, assuming $\mathcal{T}$ is countable or even finite is safe, as there is a finite smallest time increment for timestamps and a finite smallest bid increment. Moreover, the ordering policy deals with ordering transactions in a finite time interval and bids will be upper-bounded by the maximum value in the system (e.g., the maximum number of tokens). However, for the subsequent economic analysis, it will be more convenient to work with the continuum where differences in time stamps and bids can be arbitrarily small.

Having proven that score-based algorithms are essentially the only ones satisfying the independence of irrelevant transactions property, we turn to selecting the most natural one among them. Note that FCFS is the scoring function that corresponds to scoring transactions by their timestamp only while scoring transactions only by bids corresponds to the first-price auction solution. In the next section, we show how our scoring policy TimeBoost corresponds to a simple mixture of these two strategies.

8.3 TimeBoost Description

We now formally define the TimeBoost ordering policy in this section. As mentioned before, we want TimeBoost to satisfy the independence of irrelevant transactions property (i.e., it needs to be a scoring function based on Theorem 8.5) and also provide low confirmation-latency for transactions. Therefore, we will only allow TimeBoost to consider transactions within a time g interval; this granularity g can be set suitably based on the particular usecase.

Basic model. Suppose there are n transactions in the g time interval, labeled with $\text{tx}_1, tx_2, \cdots \text{tx}_n$, and sorted by increasing arrival time. Each transaction tx_i is characterized by a pair of a timestamp or arrival time, denoted by t_i, and a bid, denoted by $b_i \geq 0$. Formally, we view a transaction as a tuple of non-negative reals, $\text{tx}_i = (t_i, b_i) \in \mathbb{R}^+ \times \mathbb{R}^{\geq 0}$.

TimeBoost scoring function. Intuitively, for the TimeBoost scoring function, we propose to allow users to "buy time" using their transaction bid; in other words, transactions will be sorted by increasing timestamps (as in FCFS) but now users are allowed to decrease their effective timestamp (i.e., increase their score) through bids.

Formally, the score of a transaction $\text{tx}_i = (t_i, b_i)$ is computed as follows:

$$S(t_i, b_i) = \pi(b_i) - t_i. \tag{8.1}$$

where $\pi(b_i)$ denotes the priority or advantage gained by bidding b_i. Transactions are now chosen in descending order of their scores.

Choosing a bidding function π. To choose the bidding function π for TimeBoost, we start by defining several natural properties that should be satisfied.

1. $\pi(0) = 0$. This normalization implies that paying 0 bid gives no additional advantage.
2. $\pi'(b) > 0$ for all $b \in \mathbb{R}^+$ where π' denotes the first derivative of π with respect to the bid. This implies that the priority increases with the bid, which gives incentive to bid more for a higher priority.
3. $\lim_{b\to\infty} \pi(b) = g$. This implies that no transaction can outbid a transaction which arrived g time earlier (but any time advantage of less than g can be outbid). Through this, we can guarantee that the transaction ordering can be finalized within time g.
4. $\pi''(b) < 0$ for all $b \in \mathbb{R}^+$ where π'' denotes the second derivative of π with respect to the bid. This means that priority is concave, or equivalently, the cost of producing the (bidding) signal is convex. This is generally necessary to obtain the interior solution of the equilibrium condition.

The simplest bidding function satisfying the above constraints is the function:

$$\pi(b_i) := \frac{gb_i}{b_i + c} \tag{8.2}$$

where c is some constant. We will use this as the bidding function for TimeBoost. In the next section, we provide an economic analysis for TimeBoost. For this, we will assume that $c = 1$.

Complexity. For any incoming transaction tx $= (t, b)$, the sequencer can finalize tx after a delay of $g - \pi(b_i)$. This is because after this point, no later trans-

action can outbid tx. If transactions arrive at rate r, the space complexity of the sequencing algorithm is $\Theta(r)$ and the computational cost per transaction is $\Theta(\log r)$, assuming pending transactions are stored in a priority queue, ranked by score.

8.3.1 TimeBoost Economic Analysis Overview

We now describe the model for analyzing the economics for our TimeBoost ordering policy. The next two sections will describe this analysis in detail.

Basic model. Consider an arbitrage opportunity that occurs at some time (w.l.g., this can be taken as time 0). Users (from now on referred to as players) need to now take a decision on (1) how to send their transaction to the sequencer; this corresponds to the investment in latency; and (2) how much extra to bid for their transaction to get higher priority. We will analyze a simple economic model of this decision problem.

Assume that it costs user i the amount $c_i(t)$ to get its transaction received by the sequencer t time after the arbitrage opportunity arises. The only requirement on $c_i(t)$, for now, is that it is decreasing in increasing t. When the arbitrage opportunity arises, a player i has a valuation v_i to have its transaction included for execution the earliest, among those transactions contending for the same opportunity.

Analysis organization. We begin with an analysis with two players in Section 8.4. Within this, we consider different models based on when the latency

investment needs to occur. Broadly, we consider two models for latency investment: ex-ante (Section 8.4.1) and ex-post (Section 8.4.2). Ex-ante means that the latency investment needs to happen *before* learning the arbitrage opportunity while ex-post means that the latency investment can occur after learning about the arbitrage opportunity.

In Section 8.5, we generalize our results to many competing players.

8.4 Analysis of TimeBoost with 2 Players

As a starting point, assume that there are two players with valuations v_1 and v_2, distributed as per the cumulative distribution functions (CDFs) F_1 and F_2. That is, the probability that the valuation of player i is less or equal to x is equal to $F_i(x)$.

For each valuation v, the player may choose their specific latency investment. We can model this as a function $t_i : V \to \mathbb{R}$, such that, $t_i(v)$ is the latency / time chosen by a player i with valuation v. For simplicity, assume that the cost functions and value distributions are the same: $c_i(t) = c(t)$ and $F_i = F$. Throughout the paper, we assume that $F : [0, 1] \to [0, 1]$ is a uniform distribution with $F_i(x) = x$ iff $x \in [0, 1]$, for $i \in \{1, 2\}$, when final numerical values are derived. Obtaining numerical values for different distribution functions is very similar to that of uniform distribution, but we choose a uniform for simplicity of exposition. However, most of the computations are done for general distribution functions.

We now consider two different assumptions regarding the investment in la-

tency improvement. In the first model (ex-ante), we assume that the players need to invest in their latency infrastructure in advance: they acquire or rent servers close to the sequencer prior to knowing the value of the arbitrage opportunities they are competing for. In the second (ex-post) model, we assume that the players are able to invest in the latency after they learn about their valuation of the arbitrage. This corresponds to the case where the arbitrage opportunity itself takes some time to be realized[2]. In this case, the transaction sender can schedule its transaction through the third-party service, which guarantees the delivery of the transaction within some time interval, once the arbitrage opportunity is realized. In both cases, bidding is naturally assumed to be an interim decision, and in fact, one of the biggest advantages, as the valuation is already learned.

8.4.1 Ex-Ante Latency Investment

In this model, players learn their valuations only after they have already invested in latency infrastructure. If players can only compete through latency, the interaction between them becomes a static game. We study equilibria solutions of these games. A similar setting is considered in [232]. The results obtained in the following two subsections are concrete cases of folk results in the microeconomic theory; however, we include their proofs for completeness.

[2]Example of such an opportunity is a 12-second delay on the Ethereum network for a transaction to be scheduled.

8.4.1.1 Only latency investment

As a simple first step, we start by analyzing the game where only latency investment is allowed. Let x_i be the amount invested in latency by player i (so that he obtains a delay of $t_i(x_i)$). Let V_i denote the valuation random variable of player i. Then, player i has the following ex-ante payoff:

$$\text{Payoff}_i = \begin{cases} \mathbb{E}[V_i] - x_i & \text{if player invests strictly more than the other player} \\ \frac{1}{2}\mathbb{E}[V_i] - x_i & \text{if he invests an equal amount (assuming random tie-breaking)} \\ -x_i & \text{otherwise} \end{cases}$$

First, we note that there is no pure strategy Nash equilibrium solution of the game, in which player strategy sets consist of $\mathbb{R}^+$. It is easy to show this by case distinction on valuations: there are simple deviating strategies in each case. Next, we focus on the mixed equilibrium solution and obtain the following result.

Proposition 8.6. *There is a symmetric equilibrium in mixed strategies where each player i chooses x_i uniformly at random on the interval $(0, \mathbb{E}[V_i])$.*

Proof. By construction, the payoff of player j of playing $x_j \leq E[V_j]$ against the uniform strategy on $(0, E[V_i])$ is

$$F(x_j)E[V_j] - x_j = \frac{x_j}{E[V_i]}E[V_j] - x_j = 0.$$

Choosing a strategy $x_j > E[V_j]$ gives a negative pay-off. Therefore, each $0 \leq x_j \leq E[V_j]$ is the best response of player j, and mixing uniformly among them is also the best response.

□

The above-described equilibrium is unique up to a change of strategy on a null set and in any mixed equilibrium, both players obtain the same payoffs as in this equilibrium. Note that the result is independent of the latency cost function. The only property used is that if a player invests more than the other player in the latency technology, its transaction is scheduled earlier.

8.4.1.2 Budget constraints

We now model the fact that players may not have access to an arbitrary amount of money they need to invest to improve their latency, but are instead are constrained by a budget. Let B_i denote the budget of player i, meaning that player i cannot spend more than B_i. We consider an asymmetric case where one (weak) player has a budget $B_1 < E[V_i]$ and the other (strong) player has a larger budget with $B_2 > B_1$. First, note that similar to the previous section with unlimited access to money, there is no pure strategy Nash equilibrium. Therefore, we switch to mixed strategy equilibrium. Let F_i denote the probability distribution of playing different strategies.

Proposition 8.7. *There exists a mixed Nash equilibrium solution in the game in which the weak player receives a payoff of* 0 *and the strong player receives a payoff of* $E[V_i]-B_1$.

Proof. The following strategy profile in which the first player plays according to the following (mixed) strategy:

$$F_1(x) = \begin{cases} \frac{x}{E[V_i]} + \frac{E[V_i]-B_1}{E[V_i]}, & x \in (0, B_1], \\ \frac{E[V_i]-B_1}{E[V_i]}, & x = 0, \\ 1, & x > B_1, \end{cases}$$

the second player plays according to

$$F_2(x) = \begin{cases} \frac{x}{E[V_i]}, & x \in [0, B_1), \\ 1, & x \geq B_1, \end{cases}$$

is a mixed strategy equilibrium. The first, weak player obtains an expected payoff 0 for any choice of $0 \leq x_1 < B_1$. The second, strong player obtains an expected payoff of $E[V_i] - B_1$ for any choice $0 < x_2 \leq B_1$. Choosing $x_2 > B_1$ is wasteful for the second player and will not occur in equilibrium. Thus, both players are indifferent between all pure strategies in support of F_1 resp. F_2 and for player 2 choosing an action outside of the support of F_2 is dominated. The mixed strategies form a Nash equilibrium. □

Similarly to the unconstrained case, the above-described equilibrium is unique up to a change of strategy on a null set and in any mixed equilibrium, both players obtain the same payoffs as in this equilibrium. Also similarly to the previous section, the result is independent of the latency cost function. The only property to derive this result is that if a player invests more than the other player in the latency technology, its transaction is faster (has a lower timestamp).

8.4.1.3 Ex-ante Latency with Interim Bidding

We now analyze the model where both latency and bidding are allowed but the latency is ex-ante. That is, investment in latency happens before players learn their valuations but after learning their valuation players can use bidding to improve the transaction score.

We consider a version where players learn the other players' latency investment decisions before bidding. This models the fact that players will typically play the game repeatedly and can therefore observe latency levels of each other.

In the following let $x = (x_1, x_2)$ be the latency investment levels chosen by the two bidders and let $\Delta := t_2(x_2) - t_1(x_1)$ be the corresponding difference in latency. W.l.o.g. assume $\Delta \geq 0$. First, we consider the case that $\Delta = 0$:

Proposition 8.8. *There is a completely separating equilibrium of the bidding game when both bidders have made the same ex-ante investment.*

Proof. Given in Section 8.4.1.4 □

Next, we consider the case that $\Delta \neq 0$. For the case of different ex-ante investment we get partially separating equilibria where bidders do not bid for low valuations and bid for high valuations. The bidding strategies are asymmetric in general. However, for sufficiently large g the equilibrium becomes approximately symmetric and approximately efficient. See Figure 8.1 for a graphical illustration.

Proposition 8.9. *There is an equilibrium of the bidding game which is separating conditional on bidding: There is a threshold $\sqrt{\frac{\Delta}{g-\Delta}}$, such that a bidder does not bid if his valuation is below the threshold and bids if his valuation is above the threshold. Conditional on bidding, the high latency bidder i produces a higher signal than the low latency bidder j for equal valuations: $\pi_i(v) - t_i > \pi_j(v) - t_j$, for $v > \sqrt{\frac{\Delta}{g-\Delta}}$.*

Proof. Given in Section 8.4.1.4 □

The equilibrium analysis in Propositions 8.8 and 8.9 indicates how efficient

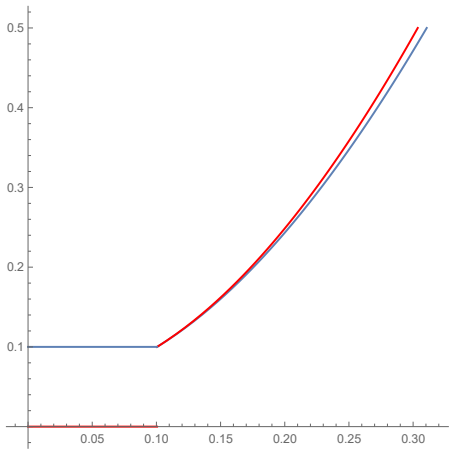

Figure 8.1: Example of equilibrium signaling functions for $g = 10$ and $\Delta = 0.1$. Timestamps are normalized so that $t_2 = 0$. The blue function is the equilibrium signal $\pi_1(v) - t_1$ for bidder 1 as a function of the valuation. The red function is the equilibrium signal $\pi_2(v) - t_2$ for bidder 2 as a function of the valuation.

our transaction ordering policy is as a function of the latency investment of bidders. If bidders have the same latency we have a standard all pay auction which yields a fully efficient outcome. If there is a difference in latency we have no bidding for low valuation bidders and approximately equal signals produced for equal valuations for high valuation bidders. Conditional on entry, low latency bidders underbid and high latency bidders overbid relative to the standard all pay strategies. Efficiency depends on the latency difference and the g parameter. If g is chosen sufficiently large the auction is approximately efficient. A too low g can be detected by low bidding activity. Hence our transaction policy can strike a balance between fairness, low latency and efficiency if properly parameterized.

8.4.1.4 Proofs

Proof of Proposition 8.8. We want to determine bidding signals $\pi_1(v_1, \Delta)$ and $\pi_2(v_2, \Delta)$, which are functions of valuations and the difference in latency. For

a given Δ denote the inverse of $\pi_1(\cdot, \Delta)$ and $\pi_2(\cdot, \Delta)$ by $\tilde{v}_1(\cdot, \Delta)$ and $\tilde{v}_2(\cdot, \Delta)$. Then bidder 1 solves at the interim stage

$$\max_{\pi \geq 0} Pr[\pi - t_1(x_1) \geq \pi_2(v_2, x) - t_2(x_2)]v_1 - \frac{\pi}{g - \pi} = F(\tilde{v}_2(\pi + \Delta, \Delta))v_1 - \frac{\pi}{g - \pi},$$

We obtain the first order condition:

$$f(\tilde{v}_2(\pi + \Delta, \Delta))v_1 \frac{\partial \tilde{v}_2(\pi + \Delta, \Delta)}{\partial \pi} = \frac{g}{(g - \pi)^2}$$

For the uniform distribution, this simplifies to:

$$v_1 \frac{\partial \tilde{v}_2(\pi + \Delta, \Delta)}{\partial \pi} = \frac{g}{(g - \pi)^2}.$$

Similarly, for bidder 2 we obtain

$$v_2 \frac{\partial \tilde{v}_1(\pi - \Delta, \Delta)}{\partial \pi} = \frac{g}{(g - \pi)^2}.$$

The two equations give a system of differential equations that need to be solved for π_1 and π_2 or alternatively for $\tilde{v}_1$ and $\tilde{v}_2$. Alternatively, we can write the system as:

$$\tilde{v}_1(\pi, \Delta) \frac{\partial \tilde{v}_2(\pi + \Delta, \Delta)}{\partial \pi} = \frac{g}{(g - \pi)^2}. \tag{8.3}$$

$$\tilde{v}_2(\pi, \Delta) \frac{\partial \tilde{v}_1(\pi - \Delta, \Delta)}{\partial \pi} = \frac{g}{(g - \pi)^2}. \tag{8.4}$$

The solution to (8.3) and (8.4) in case of equal investment (so that $\Delta = 0$) and a symmetric equilibrium is given by the following formula:

$$\tilde{v}_1(\pi, 0) = \tilde{v}_2(\pi, 0) = \sqrt{2 \int_0^{\pi} \frac{g}{(g - \pi)^2} d\pi} = \sqrt{\frac{2\pi}{g - \pi}}. \tag{8.5}$$

We solve for the signal as a function of the valuation:

$$v^2 = \frac{2\pi}{g - \pi} \Leftrightarrow \pi = \frac{gv^2}{2 + v^2}.$$

□

Proof of Proposition 8.9. When $x_1 \neq x_2$, we can first sum up (8.3) and (8.4) to obtain a differential equation for the expected payoff $v_1 v_2$:

$$\frac{d(v_1(\pi)v_2(\pi+\Delta))}{d\pi} = \frac{g}{(g-\pi)^2} + \frac{g}{(g-\pi-\Delta)^2}. \tag{8.6}$$

Integrating both sides of the differential equation above gives the solution:

$$v_1(\pi)v_2(\pi+\Delta) = \frac{\pi}{(g-\pi)} + \frac{\pi+\Delta}{g-\pi-\Delta} + K. \tag{8.7}$$

To determine the constant we need to determine boundary conditions. For bidder 1, at the threshold where he is indifferent between bidding and not bidding, we have $\pi_1 = 0$ and for bidder 2, at the threshold where he is indifferent between bidding and not bidding, he needs to overcome the handicap, we have $\pi_2 = \Delta$. At the threshold bidder 2 should make the same profit as from pooling,

$$v_1(0)v_2(\Delta) = \frac{\Delta}{g-\Delta} \Rightarrow K = 0.$$

Combining (8.7) and (8.4) we obtain a separable differential equation:

$$\frac{dv_1(\pi,\Delta)}{v_1(\pi,\Delta)} = d\pi \frac{g}{(g-\pi-\Delta)^2}\left(\frac{\pi}{g-\pi} + \frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1}. \tag{8.8}$$

Combining (8.7) and (8.3) we obtain another separable differential equation:

$$\frac{dv_2(\pi+\Delta,\Delta)}{v_2(\pi+\Delta,\Delta)} = d\pi \frac{g}{(g-\pi)^2}\left(\frac{\pi}{g-\pi} + \frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1}. \tag{8.9}$$

Integrating both parts of the equation (8.8) solves the (logarithm of) the value as a function of the bid:

$$\ln(v_1(\pi)) - \ln(v_1(0)) = \int_0^{\pi} \frac{g}{(g-\pi-\Delta)^2}\left(\frac{\pi}{g-\pi} + \frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi.$$

Similarly, integrating both parts of the equation (8.9) solves the (logarithm of) the value as a function of the bid:

$$\ln(v_2(\pi+\Delta)) - \ln(v_2(\Delta)) = \int_0^{\pi} \frac{g}{(g-\pi)^2}\left(\frac{\pi}{g-\pi} + \frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi.$$

To determine the marginal valuations $v_1(0)$ and $v_2(\Delta)$ at which the two bidders start bidding, note that the support of $\pi_i - t_i$ and that of $\pi_j - t_j$ need to coincide for valuations where we have separation of types. Therefore, $v_1(0) = v_2(\Delta)$. Since $v_1(0)v_2(\Delta) = \frac{\Delta}{g-\Delta}$ it follows that $v_1(0) = v_2(\Delta) = \sqrt{\frac{\Delta}{g-\Delta}}$. This is the threshold where bidders start bidding. It follows that for $\Delta \neq 0$

$$v_1(\pi) = \sqrt{\frac{\Delta}{g-\Delta}} \exp\left(\int_0^{\pi} \frac{g}{(g-\pi-\Delta)^2}\left(\frac{\pi}{g-\pi} + \frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi\right)$$

and

$$v_2(\pi) = \sqrt{\frac{\Delta}{g-\Delta}} \exp\left(\int_0^{\pi-\Delta} \frac{g}{(g-\pi)^2}\left(\frac{\pi}{g-\pi} + \frac{\pi+\Delta}{g-\pi-\Delta}\right)^{-1} d\pi\right).$$

To compare the equilibrium signals $\pi_1(v) - t_1$ and $\pi_2(v) - t_2$ for $v > \sqrt{\frac{\Delta}{g-\Delta}}$, we need to compare $\pi_1(v) + \Delta$ to $\pi_2(v)$.

From the expressions for the valuations as a function of the bid, we can observe (observe that $\frac{g}{(g-\pi-\Delta)^2} \geq \frac{g}{(g-\Delta)^2}$) that

$$v_1(\pi) > v_2(\pi + \Delta),$$

for $\pi > 0$. It follows that

$$\pi_1(v) \leq \pi_2(v) - \Delta,$$

for $v \geq \sqrt{\frac{\Delta}{g-\Delta}}$. □

8.4.2 Ex-Post Latency with Bidding

We now analyze the ex-post model with bidding; here both the latency investment, and the bid can be made after the valuation is observed. First, we start

with only the latency investment decision. The expected utility of player i is equal to:

$$Pr[t(v_i) < t(v_j)]v_i - c(t(v_i)),$$

where $j \in \{1, 2\} \setminus i$.

We can look at this from a dual perspective: by $v(t)$ we define the inverse of $t(v)$. This is the so-called *Revelation Principle*. Instead of some function of the type, we report our type directly. Then, the optimization problem becomes:

$$\max_v Pr[v \geq v_2]v_1 - c(t(v)). \tag{8.10}$$

By replacing the probability with $F(v)$, we get that it is equivalent to

$$\max_v F(v)v_1 - c(t(v)).$$

By the first order condition, we get:

$$v_1 f(v) - c'(t(v))t'(v)|_{v=v_1} = 0,$$

where f is a density function of the valuation distribution F. By plugging in $v = v_1$, it is equal to:

$$v_1 f(v_1) - c'(t(v_1))t'(v_1). \tag{8.11}$$

For the uniform distribution and cost function $c = \frac{1}{t}$, first order condition gives the following differential equation:

$$v_1 + \frac{t'(v_1)}{t^2(v_1)} = 0. \tag{8.12}$$

Solving this equation gives $t(v) = \frac{2}{c_1+v^2}$. By the boundary condition that 0 valuation type should wait infinitely (or equivalently pay 0 in the latency), we obtain the value of the constant in the solution: $c_1 = 0$. Therefore, cost incurred is equal to $\frac{1}{t} = \frac{v^2}{2}$. On average each player pays:

$$\int_0^1 \frac{v^2}{2} f(v)dv|_0^1 = \frac{1}{6},$$

for better latency, before learning their types. The cost of producing score $s = \frac{gm}{m+1} - t$ is:

$$c(s) := m + \frac{1}{t}. \tag{8.13}$$

We decompose total expenditure into 2 parts, for bidding and for time, by representing m and $c(t(v))$ as functions of v and taking integrals:

$$b(g) := \int_0^1 m(v)f(v)dv \text{ and } \int_0^1 \frac{1}{t(v)} f(v)dv.$$

Proposition 8.10. *The limit of $b(g)$ when g tends to infinity is equal to $\frac{1}{6}$. $b(g)$ is an increasing function in g, for g large enough.*

Proof. Given in Section 8.4.2.1 □

The proposition implies that by taking large enough g, the system extracts almost all value invested in the latency through bidding. Starting from some threshold value on g, extraction increases with increasing g.

We can verify whether the constructed equilibrium is unique by checking the conditions given in [184].

Example 8.11. We can calculate a few values of $b(g)$. In particular, $b(1000) \approx 0.1294$, meaning a player pays approximately 77% of the total expenditure in bids, and $b(10000) \approx 0.1537$, meaning a player pays approximately 92% of the total expenditure in bids.

Note that in the proof of the proposition 8.10, the total investment in both latency and bidding, $c(v)$, is the same value $\frac{v^2}{2}$, as in the case of only investing in the latency. We show that this is not a coincidence. In general, assume that there is an arbitrary signaling technology described by an increasing, differentiable cost function $C(s)$. The following result shows the revenue equivalence of ex-post bidding:

Proposition 8.12. *Both players spend the same amount on average for any cost function C.*

Proof. Given in Section 8.4.2.1 □

The amount spent depends only on the value belief distribution function.

8.4.2.1 Proofs

Proof of Proposition 8.10. The optimization problem of the player in the equilibrium is to minimize cost, subject to the score equation constraint. By plugging in $t = \frac{gm}{m+1} - s$, we obtain the minimization problem:

$$\min_m \left(m + \frac{m+1}{gm - s(m+1)} =: x(m) \right).$$

The first order condition on $x(m)$ gives:

$$\frac{dx(m)}{dm} = 1 + \frac{gm - s(m+1) - (m+1)(g-s)}{(gm - s(m+1))^2} = 1 - \frac{g}{(gm - s(m+1))^2} = 0, \tag{8.14}$$

gives that the value of m that minimizes the cost function. The solutions of the last equation are $gm - sm - s = \sqrt{g}$ equivalent to $m = \frac{s+\sqrt{g}}{g-s}$ and $gm - sm - s = -\sqrt{g}$ equivalent to $m = \frac{s-\sqrt{g}}{g-s}$, or the boundary condition $m = 0$. For $m = 0$, the value $x(0) = -\frac{1}{s}$, while for $m = \frac{s+\sqrt{g}}{g-s}$, the value

$$x\left(\frac{s+\sqrt{g}}{g-s}\right) = \frac{s+\sqrt{g}}{g-s} + \frac{\frac{s+\sqrt{g}}{g-s} + 1}{g\frac{s+\sqrt{g}}{g-s} - s(\frac{s+\sqrt{g}}{g-s} + 1)} = \frac{1 + 2\sqrt{g} + s}{g-s}.$$

Accordingly, the marginal cost of producing signal s is:

$$c'(s) = \begin{cases} \frac{(1+\sqrt{g})^2}{(g-s)^2}, & \text{if } s > -\sqrt{g}, \\ \frac{1}{s^2}, & \text{if } s \leq -\sqrt{g}. \end{cases}$$

We solve a similar differential equation as (8.11), just with different marginal cost function c', and instead of time function t, we have a score function s of valuation v. The differential equation becomes:

$$vf(v) - c'(s)s'(v) = 0. \tag{8.15}$$

We need to solve for the $s(v)$ function. For types v with $\frac{2}{v^2} \geq \sqrt{g}$ who only use latency we have the same solution as before

$$s(v) = -\frac{2}{v^2}.$$

The marginal type who is indifferent between using only latency and using a combination of the two technologies is given by

$$u = \sqrt{\frac{2}{\sqrt{g}}}.$$

which gives the boundary condition $s(u) = -\sqrt{g}$ for the differential equation describing the behavior of types who choose a signal $s \geq -\sqrt{g}$:

$$v = \frac{(1+\sqrt{g})^2}{(g-s)^2} s'(v).$$

We obtain the solution

$$s(v) = (4c_1 g^{3/2} + 2c_1 g^2 + 2c_1 g + g(v^2-2) - 4\sqrt{g} - 2)/(2c_1 g + 4c_1\sqrt{g} + 2c_1 + v^2). \quad (8.16)$$

The value of the constant c is obtained from the boundary condition that a zero-value player does not invest and it is equal to

$$c_1 = \frac{1}{(1+\sqrt{g})^2}.$$

Therefore, plugging in the constant value in the solution (8.16) and simplifying it gives:

$$s(v) = \frac{gv^2 - 4\sqrt{g} - 2}{v^2 + 2}.$$

Plugging this into the formula of $c(s)$, gives the cost value as a function of valuation v:

$$c(v) = \frac{1 + 2\sqrt{g} + \frac{gv^2 - 4\sqrt{g} - 2}{v^2+2}}{g - \frac{gv^2 - 4\sqrt{g} - 2}{v^2+2}} = \frac{v^2}{2}.$$

Separate expenditure in the bidding is calculated by the following formula:

$$b(g) = \int_u^1 m(v)f(v)dv = \int_{\sqrt{\frac{2}{\sqrt{g}}}}^1 \frac{\frac{gv^2-4\sqrt{g}-2}{v^2+2} + \sqrt{g}}{g - \frac{gv^2-4\sqrt{g}-2}{v^2+2}}dv =$$
$$\int_{\sqrt{\frac{2}{\sqrt{g}}}}^1 \frac{v^2(g+\sqrt{g}) - 4\sqrt{g} - 2}{2g - 4\sqrt{g} - 2}dv =$$
$$\frac{1}{2g + 4\sqrt{g} + 2}\left(\frac{g+\sqrt{g}}{3}(1 - \frac{2}{\sqrt{g}}\sqrt{\frac{2}{\sqrt{g}}}) - (4\sqrt{g}+2)(1 - \sqrt{\frac{2}{\sqrt{g}}})\right).$$

The dominant term in the nominator above is g and also in the denominator, it is $6g$. Therefore, $\lim_{g\to\infty} b(g) = \frac{1}{6}$.

□

Proof of Proposition 8.12. We are interested in the equilibrium signaling strategy $s(v)$. Suppose that this strategy is increasing (so no pooling of types) and differentiable. Then, we can define a differentiable function

$$\tilde{C}(v) := C(s(v)).$$

To figure out what $\tilde{C}(v)$ is, we have to consider an optimization problem with the first player:

$$\max_v Pr[v \geq v_2]v_1 - C(s(v)) = Pr[v \geq v_2]v_1 - \tilde{C}(v).$$

Taking first order conditions with respect to v gives:

$$v_1 f(v) - \tilde{C}'(v)|_{v=v_1} = 0,$$

that is,

$$v_1 f(v_1) = \tilde{C}'(v_1).$$

For the uniform distribution:

$$v_1 = \tilde{C}'(v_1).$$

Using the boundary condition $\tilde{C}(0) = 0$ and integrating we get

$$\tilde{C}(v_1) = v_1^2/2.$$

More generally:

$$\tilde{C}(v_1) = \int_{-\infty}^{v_1} v f(v) dv.$$

□

8.5 Analysis of TimeBoost with n players

In this section, we consider n players with the same valuation distribution as in the 2 players case. The optimization problem is now the following:

$$\max_v Pr[v \geq \max\{v_2, \cdots, v_n\}]v_1 - c(t(v)),$$

similarly to (8.10). By replacing the probability with cumulative distribution, this is equivalent to:

$$\max_v F_{n-1}(v)v_1 - c(t(v)),$$

where $F_{n-1}(x)$ is a cumulative distribution function of the random variable $X := \max\{X_1, \cdots, X_{n-1}\}$. By independence we have

$$F_{n-1}(x) = F(x)^{n-1}.$$

The first-order condition and plugging in $v = v_1$ gives the following differential equation, similar to (8.11):

$$f_{n-1}(v_1)v_1 - c'(t(v_1))t'(v_1) = 0,$$

where $f_{n-1}(v_1) = (n-1)v_1^{n-2}$ is a density function of maximum among $n-1$ uniformly distributed random variables. The differential equation w.r.t. $t(v)$ becomes:

$$(n-1)v_1^{n-1} + \frac{t'(v_1)}{t^2(v_1)} = 0.$$

Solving the equation gives $t(v) = \frac{n}{c+(n-1)v^n}$. The same boundary condition ensures that $c = 0$, that is, $t(v) = \frac{n}{(n-1)v^n}$. Each player pays:

$$\frac{n-1}{n}\int_0^1 v^n dv = \frac{n-1}{n}\frac{v^{n+1}}{n+1}|_0^1 = \frac{n-1}{n(n+1)}.$$

Together, the players pay $\frac{n-1}{n+1}$, that converges to 1 as n converges to infinity. Note that the first place in the transaction order is given to the maximum-value player. The average valuation of the maximum value player is $\frac{n}{n+1}$, order statistics. This value also converges to 1 as n tends to infinity.

The analysis is the same as in the case of 2 players, until the differential equation that solves score function s. Instead of (8.15), for n players we solve:

$$(n-1)vv^{n-1} - c'(s)s'(v) = 0. \tag{8.17}$$

For types v with $\frac{n}{(n-1)v^2} \geq \sqrt{g}$, who only use latency, we have the same solution as before

$$s(v) = -\frac{n}{(n-1)v^2}.$$

Marginal type investing in bidding is:

$$u = \sqrt{\frac{n}{(n-1)\sqrt{v}}}.$$

Plugging in functional forms of c and s in (8.17) gives the same limit results as in Proposition 8.10. Next, we show a revenue equivalence for n players. The

argument is similar to 2 players' case. Assume that there is an arbitrary signaling technology described by an increasing, differentiable cost function $C(s)$.

Proposition 8.13. *All n players spend the same amount on average for any cost function C.*

Proof. We are interested in the equilibrium signaling strategy $s(v)$. Suppose that this strategy is increasing (so no pooling of types) and differentiable. Then, we can define a differentiable function

$$\tilde{C}(v) := C(s(v)).$$

To figure out what $\tilde{C}(v)$ is, we have to consider an optimization problem of the first player:

$$\max_v Pr[v \geq \max\{v_2, \cdots, v_n\}]v_1 - \tilde{C}(v) = F(v)^{n-1}v_1 - \tilde{C}(v).$$

Taking first order conditions with respect to v:

$$\left[(n-1)v_1 f(v)F(v)^{n-2} - \tilde{C}'(v)\right]\Big|_{v=v_1} = 0,$$

For the uniform distribution, we get:

$$(n-1)v_1^{n-1} = \tilde{C}'(v_1).$$

Using the boundary condition $\tilde{C}(0) = 0$ and integrating we get

$$\tilde{C}(v_1) = \frac{(n-1)v_1^n}{n}.$$

More generally:

$$\tilde{C}(v_1) = \int_{-\infty}^{v_1} (n-1)vf(v)F(v)^{n-2}dv.$$

□

8.6 Comparison of TimeBoost with a Pure Bidding Policy

We now compare TimeBoost to what to a pure bidding policy. Recall that for the bidding policy, all transactions sent in fixed time intervals of length g are collected, and sorted in decreasing order of their bids. This effectively simulates a first-price all-pay auction for each interval. We note this can be thought of as a quantized version of TimeBoost, because it produces the same sequence that would be produced by first rounding off each transaction's arrival timestamp to the nearest multiple of g and then applying TimeBoost.

Generically speaking, a first-price auction where only the winning bidder pays and first-price all-pay auctions are both *payoff equivalent* for Bayesian-Nash incentive compatible mechanisms, (see e.g., [195]). In our setting, the following result holds for each individual arbitrage opportunity.

Proposition 8.14 (see [195])**.** *The expected payoff of the bidding game where the only the highest bidder pays their bid is equal to the expected payoff in the bidding game where the highest bidder wins but all players pay their bids, independently of valuation distributions.*

For simplicity, to compare TimeBoost with a pure bidding policy, we consider two players. It is straightforward to generalize to more parties. For a given arbitrage opportunity, two cases arise as described below depending on whether transactions can be submitted within the same g-time interval as the arbitrage opportunity or not:

1. Both players can submit their transactions within the same g interval. For the pure bidding policy, if both players can get their transaction submitted

inside the same g-time interval as the arbitrage opportunity, then they will both compete for it. It is easy to see that when the valuations of the two parties are the same, the bidding strategy for the pure-bidding policy vs the ex-ante latency with bidding policy will be the same. In other words, in this scenario, TimeBoost maintains the economic properties of the first-price auction pure-bidding policy.

2. Only one player can get its transaction within the same g interval. If only one player can get its transaction inside the same g-time interval as the arbitrage opportunity, then in the pure-bidding policy, that player can pay a 0 bid and still take advantage of it. In contrast, since TimeBoost does not require discrete boundaries, both players will always have g time to submit their transactions (recall that bidding can be used to get priority over any transaction received up to g time earlier). This means that even for a reasonably small g (say 0.5 sec), both parties will always be able to compete for the opportunity. In equilibrium, this results in bids equal to value of the arbitrage.

Analysis for the second case. Suppose the first party (denoted by A) can reach the sequencer in s_1 time, and the second party (denoted by B) can reach in s_2 time, with $s_1 < s_2$. Then, with the pure-bidding policy, A can wait until $g - s_1$ seconds pass since the beginning of a new block creation, and send its transaction to the sequencer at exactly $g - s_1$, while B has to send its transaction by time $g - s_2$ in order to be included in the same block.

Assuming that arbitrage opportunities are uniformly distributed over the g-interval, this means that, with probability $\frac{g-s_1-(g-s_2)}{g} = \frac{s_2-s_1}{g}$, B has no chance to win the race against A, even if it values the arbitrage opportunity much more

than A. When g is large (e.g., on Ethereum with 12 sec block times), this latency advantage is not a big issue, as A would only have an advantage with probability $(s_2 - s_1)/12$. In contrast, for faster blockchains, or layer-2 rollups which have shorter block-times to achieve scalability, this latency advantage can be significantly more important in the pure-bidding policy vs in TimeBoost. For instance, when $g = 0.5$sec, A's latency advantage is 24 times greater than what it was in Ethereum. This means that compared to TimeBoost, a pure-bidding strategy will either result in substantial latency competition (when g is small) or will not be able to provide low transaction finalization time (since g will be large).

8.7 Discussion on Sequencer Decentralization

We now briefly discuss how TimeBoost can be supported by a decentralized sequencer—i.e., a committee of ℓ sequencers (of which at most some f can be dishonest). We only provide possible implementations here; a formal rigorous analysis is outside the scope of this paper.

In the decentralized setting, transactions to be sequenced are now submitted by users to all sequencers instead of just one. Note that as before, threshold decryption techniques can be used for transaction privacy before ordering.

The most natural way to support TimeBoost in a decentralized setting is to have a protocol for sequencers to agree on both the timestamp and the bid of transactions. After this is done, the TimeBoost policy can simply be applied on the consensus output of the decentralized committee to obtain the final ordering. Agreeing on the bid is easy since we can have the same bid be submitted to

all sequencers for a given transaction. Agreeing on the timestamp is a more challenging problem since the same transaction can arrive at different nodes. While it adds significant complexity, one potential technique here is to employ a fair-ordering protocol (this can be as a simple as e.g., computing the median timestamp [259, 170] or support more complicated techniques as in [161, 158, 54]). We leave the formal analysis of such a decentralized TimeBoost implementation to future work.

8.8 Conclusion

We designed TimeBoost: a policy for transaction ordering that takes into account both transaction arrival times and bids. We showed that any ordering scheme that guarantees the independence of different latency races is a generalized scoring rule. By choosing a suitably designed mixture of timestamps and bids, we showed the economic efficiency of the system: transaction senders spend most of their resources on bidding instead of latency improvement, which can later be used by the protocol for improvement and development.

Part II

Insights from New Attacker Capabilities

CHAPTER 9

PROOFS OF COMPLETE KNOWLEDGE

9.1 Introduction

Most cryptographic protocols are designed under a simple model of knowledge. If a player $\mathcal{P}$ can explicitly furnish a secret value sk as a (private) protocol input, then she *knows* sk. In a digital signature scheme, for example, $\mathcal{P}$ inputs a private key sk to a locally executed algorithm to sign a message.

This basic, intuitive model of knowledge, however, can break down when sk is not controlled by a single player, but an *interactive functionality*. For example, sk might be stored exclusively in a trusted execution environment (TEE) such as Intel SGX [188, 187], AMD SEV [7], or AWS Nitro Enclaves [6]. The TEE could then *encumber* $\mathcal{P}$'s access to sk, by only allowing selective use. For instance, in the previous digital signatures example, a TEE could generate (and store) a private signing key sk for a user Alice, but only allow Alice to sign messages approved by an adversary.

Such encumbrances can also be realized by multi-party computation (MPC) [135, 255] over sk among a committee that restricts its use. Encumbrance of secrets can undermine security in many cryptographic protocols, as we will see.

It may seem counterintuitive that a user / prover might *want* to encumber her own secret sk. Encumbrance of secrets, it turns out, can paradoxically *benefit* a user. For example, as highlighted in [93, 215], voters that choose to encumber secret keys used in a voting scheme can sell their votes to an adversary trying

to subvert an election. Here, Alice might encumber her voting key sk so that she can only sign a ballot with candidate Bob, the choice of adversary Mallory. Alice can then sell Mallory an enforceable promise that if she votes, she will vote only for Bob. Remarkably, even techniques that specifically aim to prevent such vote-selling—e.g., so-called coercion-resistant voting schemes [82, 83, 154]—fail in the presence of such key encumbrance.

This chapter introduces and explores a new notion of knowledge called *complete knowledge* (CK). Complete knowledge embodies a strong notion of possession meant to rule out encumbrance of e.g., the secret key. CK by a prover $\mathcal{P}$ of a secret sk means, informally, that it has *unencumbered* access to sk and *can use it for any desired purpose*, e.g., can sign any message of her choice.

CK can be leveraged in our voting example, by requiring Alice to prove that she has complete knowledge of her secret key sk before allowing her to vote. Here, CK would imply that Alice can always cast any desired vote and therefore, cannot sell Mallory an enforceable promise to vote only for Bob.

Our goals in this work are to formalize CK, implement CK schemes end-to-end, and shed light on its various applications.

The problem with proofs of knowledge. To understand CK, we build on the classical formalism of *proofs of knowledge* (PoKs). PoKs are interactive protocols in which a prover demonstrates knowledge of some kind to a verifier. PoKs play an important role in many cryptographic constructions and have found widespread applications in e-voting [70, 71, 83, 186], encryption [217, 224], group signatures [16], and private cryptocurrency transactions [49, 235].

More formally, a PoK involves two players: a *prover* $\mathcal{P}$ and a *verifier* $\mathcal{V}$. The goal is for $\mathcal{P}$ to convince an honest $\mathcal{V}$ that it knows a valid *witness* sk for some (public) statement x. In practice, PoKs are often *zero-knowledge* [116], meaning that the protocol hides any information about sk from $\mathcal{V}$. (A PoK need not be zero-knowledge, though, and this is true of some CK schemes we propose.)

As observed above, however, $\mathcal{P}$ could have access to sk intermediated by an interactive protocol with another entity or device (e.g., a TEE or an MPC committee). In this case, classical PoK formalism breaks down, because it lacks a notion of encumbered knowledge.

In our voting example above, for instance, Alice has a secret key sk encumbered in a TEE that only allows her to cast a vote for Bob in an election. This same TEE, however, might allow unencumbered use of sk for completely different purposes, e.g., signing to authorize cryptocurrency transactions. In these contexts, Alice along with the TEE *can successfully prove knowledge* of sk. Yet *neither Alice nor any other entity truly knows* sk, in the sense of being able to use it for *any* desired purpose. In fact, the TEE can allow Alice to take any action using sk *except for the one to vote against Bob*. This allows a powerful collusion between Alice and the bribing adversary Mallory; Mallory can bribe Alice without risk since she provably cannot vote against Bob, while Alice is willing to accept the bribe since Mallory learns no information about the key and at the same time Alice is guaranteed to be able to utilize her key for all other functionalities.

This situation underscores a mismatch between the existing formalism for proofs of knowledge and knowledge in a critical, real-world sense. In this paper, we show the potential practical impact of this mismatch, describing coercive attacks that exploit encumbrance of secret keys in voting schemes, deniable

messaging, and blockchain systems.

We show how to remedy the resulting problems by introducing the notion of CK.

Proofs of complete knowledge (PoCKs). In this work, we introduce and formalize proofs of complete knowledge (PoCKs). (Our definitions also cover *arguments* of complete knowledge (ACKs), for which $\mathcal{P}$ is polynomial time. We often informally use the term PoCK to denote both.)

The core idea in our formalization is intuitively as follows: A PoCK scheme ensures complete knowledge if it is the case that when an honest $\mathcal{V}$ accepts a proof by $\mathcal{P}$, $\mathcal{P}$ *can learn her own witness* sk *fully during the proof execution*.

Specifically, in a PoCK, $\mathcal{P}$ must be able to *eavesdrop* on an unencrypted channel carrying sk. To ensure this "self-eavesdropping" capability, $\mathcal{P}$ is given access to a special *resource* $\mathcal{R}$ required for successful execution of the proof.

$\mathcal{R}$ will typically be a local piece of hardware within the trust domain of $\mathcal{P}$. $\mathcal{R}$ may paradoxically itself be a TEE that stores sk—as a way of preventing encumbrance using another TEE. Alternatively, $\mathcal{R}$ could be a resource, such as an ASIC, with special computational capabilities. Informally, an *eavesdropper* $\mathcal{E}$ must be present on the channel between $\mathcal{P}$ and $\mathcal{R}$. Abstractly, $\mathcal{E}$ can be visualized as the *physical manifestation* of a straight-line extractor [122] from PoK literature, in the sense that it will allow for extraction in practice rather than just as a proof construct (see Section 9.4 for details).

In the case of $\mathcal{R}$ being a TEE, $\mathcal{E}$ can be constructed by simply requiring $\mathcal{P}$ to submit sk in plaintext to $\mathcal{R}$, or alternatively by having a function within the

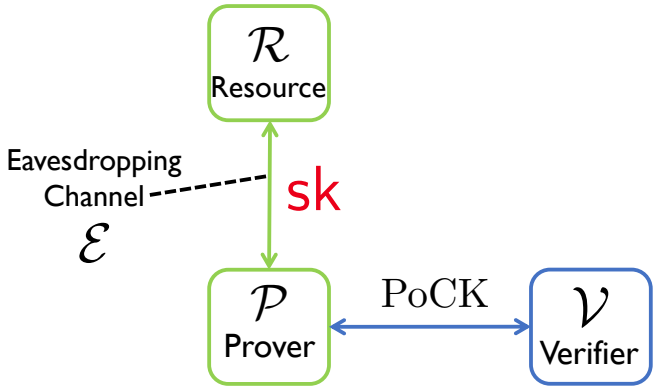

Figure 9.1: *Proof of complete knowledge* (PoCK) setting. A PoCK is a proof of knowledge, but has two additional requirements: (1) $\mathcal{P}$ can only execute the proof successfully by accessing a special local *resource* $\mathcal{R}$; and (2) $\mathcal{P}$ must have access to the witness sk as it is transmitted to $\mathcal{R}$ over a plaintext *eavesdropping channel* $\mathcal{E}$. Green lines / boxes in the figure indicate entities local to / within the trust domain of $\mathcal{P}$, while blue lines / boxes indicate those outside.

TEE application that reveals sk. Even if $\mathcal{P}$ uses a TEE or MPC committee in an attempt to encumber sk—thus hiding sk from herself—exposure on $\mathcal{E}$ means that $\mathcal{P}$ can still recover sk. Thus a PoCK ensures that sk is unencumbered. The PoCK setting is shown in Figure 9.1.

Let's return to our voting example now. Intuitively, if a PoCK is used here, then Mallory will no longer have any guarantee on Alice's vote since Alice can use $\mathcal{E}$ to fully recover sk without being detected by Mallory.

Realizing CK. We consider two practical schemes for realizing PoCKs. These schemes enable any proof of knowledge protocol to be converted into a PoCK protocol.

Our first scheme is a conceptually straightforward one that generalizes an idea proposed by Gunn et al. [144] for preventing TEE-based attacks on denia-

bility in messaging protocols. This scheme paradoxically realizes the resource $\mathcal{R}$ as a TEE *that prevents encumbrance* by, e.g., another TEE. The idea is that $\mathcal{P}$ generates sk inside or inputs sk to a special TEE application that outputs sk to the user on demand, thus realizing an eavesdropping channel $\mathcal{E}$. To prove complete knowledge of sk, $\mathcal{P}$ has the special TEE application generate an attestation proving that sk is output to $\mathcal{P}$. We prove the security of this CK scheme under the assumption that it is not practical to run a TEE instance inside another TEE instance.

TEEs have important drawbacks, though. Notably, one widely available, fully-featured hardware TEE is Intel SGX, in which a number of serious vulnerabilities have been discovered (and later patched), e.g., [44, 241, 242].[1]

We thus explore a second PoCK realization involving *proof of work* (PoW) [106, 152], using an ASIC as the resource $\mathcal{R}$.

An ASIC (Application-Specific Integrated Circuit) is special-purpose hardware for a specific computation. ASICs are widely used for cryptocurrency mining [238], specifically hashing (e.g., double SHA-256 for Bitcoin) and thus widely available as an off-the-shelf component for our PoCK scheme. ASICs also have an important feature for our purposes: they accept only *unencrypted* inputs.[2]

Our ASIC-based PoCK requires $\mathcal{P}$ to solve a specially crafted PoW puzzle within a certain time period. Successfully solving this puzzle will reveal sk to

[1] While a break of SGX (or other TEEs) may seem to *help* achieve CK by exposing sk, it can in fact *undermine* CK proofs by enabling the generation of fake TEE attestations. At the same time, a TEE break does not necessarily prevent encumbrance of secrets, as the adversary who has broken a TEE may be distinct from one who is using the TEE to encumber secrets. Also, there are alternative ways to encumber secrets, e.g., MPC.

[2] Even if an ASIC supported encrypted connection, without a built-in enclave, it would still be eavesdroppable.

$\mathcal{P}$ with high probability. Complete description of our scheme and its security analysis is given in Section 9.6.

Interestingly, our approach to CK not only contributes to the theory of proof systems, but draws on techniques from the literature on this theory, specifically the *straight-line, non-programming extractor* construction of Fischlin [122].

To show practicality of our CK schemes, we implement an end-to-end system SMACK (Section 9.7) in which the verifier is a smart contract on the Ethereum blockchain. This implementation both shows how minimal a CK verifier can be and supports the blockchain applications we discuss in Section 9.2. The ASIC-based implementation (Section 9.7.1) uses an off-the-shelf mining ASIC. In order to make CK widely accessible, we also implemented our TEE-based scheme as a mobile app (Section 9.7.2) using the mobile phone's TEE.

Contributions. In brief, our contributions are:

- *Complete Knowledge:* We introduce a new notion of knowledge for cryptographic protocols, called *complete knowledge* (CK), that addresses theoretical and practical limitations of classical proofs of knowledge arising from encumbrance of secrets.
- *CK applications:* We revisit known attacks on existing protocols that lack CK and offer a unified treatment and discussion of countermeasures through the lens of CK (Section 9.2). We also introduce two new applications of CK that help prove ownership of blockchain assets.
- *Formalization:* We formalize CK as a strengthening of standard proofs / arguments of knowledge (Section 9.3 and 9.4).

- *Practical CK schemes:* We present two practical schemes to convert proofs of knowledge to proofs of complete knowledge (PoCKs), using a TEE (Section 9.5) or off-the-shelf mining ASICs (Section 9.6). Our schemes work for a very broad class of Σ-protocols, a common type of ZK proof of knowledge in practice. *We implement both our CK schemes in a full, end-to-end system in which verification is performed in a smart contract (Section 9.7).*

9.2 Motivation: Attacks & Applications

To motivate our exploration, we begin by briefly reviewing three key-encumbrance attacks from existing literature and explain how CK can serve as a countermeasure. The result is a new, unified treatment of these attacks that identifies their common root cause as a lack of CK for private credentials. We also describe two *new applications* enabled by CK; both relate to blockchains.

9.2.1 Attacks and CK-Based Countermeasures

Authentication protocols that lack CK are vulnerable to various attacks in which credentials, such as secret keys or passwords, are made available fully or conditionally to an adversary. Three concrete examples are described below.

Deniable Messaging: Deniability in messaging protocols is the (desirable) property that a participant Alice cannot prove the authenticity of a transcript of her communication with another participant Bob, to an outsider, rendering their communications inadmissible as evidence against Bob. Several widely de-

ployed messaging protocols, e.g., Signal [233, 86] (which is based on OTR [43]), advertise such deniability as a key feature. These protocols accomplish deniability by exposing key material to two communicating parties, Alice and Bob, that allows either of them to forge a transcript unilaterally. E.g., Alice can do so using her long-term private key A and an ephemeral private key a.

Gunn et al. [144] show how a TEE can erode the deniable authentication [43, 101] that underpins messaging protocols. The use of a TEE enables a simple attack on deniability by *either* communicating party in isolation. It suffices for Alice, for example, to generate her ephemeral private key a in a special TEE application that does not permit her to forge messages from Bob. Through this technique, Alice can show a judge that she lacks the ability to forge making it so that Bob loses deniability for the messages he sends; notably this happens *without Bob realizing that his deniability has been eroded*.

Gunn et al. describe some countermeasures, the simplest counter-intuitively involving use of a small TEE program that attests that a user's private key is present in unprotected memory, i.e., *outside* the enclave, and thus made available to the user for transcript forgery.

Our work, by introducing CK, formalizes and extends ideas which allow the countermeasure from [144] to work.

Electronic Voting: Electronic voting is becoming important in decentralized systems such as permissionless blockchains [186, 162]. Those systems make use of user-generated keys. They are vulnerable to attacks—like those initially described in [93] and subsequently in [215]—in which a voter encumbers her private key used for voting in a TEE upon generation. The voter then offers a

briber or coercer *exclusive and verifiable access* to her key either permanently or for certain elections. Access is verifiable because the TEE can present a proof of encumbrance to the adversary. Note that the key can be encumbered for specific well defined tasks and does not for example, compromise the voter's cryptocurrency if the same key also controls her cryptocurrency.[3]

CK offers a countermeasure to such attacks. If a voter is required to prove CK for her private key sk, then she cannot encumber it. While prior literature has proposed *coercion-resistant* (which subsumes receipt-free [29]) voting protocols [83, 154, 176] for the explicit purpose of countering adversarial coercion, we emphasize that they are insecure against TEE-based *encumbrance attacks* [215]. CK is a *necessary* requirement to restore coercion-resistance in these protocols under encumbrance attacks.

Coercion-resistant protocols involve either: (1) An authority that sends keys to voters over an untappable channel, along with the ability of users to present fake versions of these keys to adversaries or (2) The ability for the voter to re-vote. CK *cannot* fully remedy the problem on its own for approach (1), as the TEE can still identify true keys to adversaries. But CK *can* at least prevent an adversary from gaining *exclusive* access to voting keys and *can* restore the coercion-resistance for protcols using approach (2). We leave it as an open problem to understand whether coercion-resistance can be easily restored in the first approach[4].

[3]Because a TEE can be taken offline, encumbrance in a TEE alone only ensures a briber that either the briber will be able to cast a vote or no one will. While limited, this property is still valuable to the adversary. Networks of TEEs, e.g., [93, 199] can in fact further help ensure liveness.

[4]One potential approach is to use CK along with placing all voter functionality in a TEE application. This works in theory but comes at the cost of an application-specific realization and bloated trusted-computing base.

Similarly, CK offers a path to shoring up security in *minimal anti-collusion infrastructure* (MACI), a scheme proposed by Buterin [52] that is designed to provide bribery-resistance in voting and other applications in a way loosely analogous to coercion-resistance [29, 83, 154, 176].[5] Recent strides have been made toward MACI deployment on Ethereum [153]. As explicitly noted in [52], MACI is vulnerable to TEE-based encumbrance of keys. Use of CK, however, can restore the scheme's bribery-resistance properties.

9.2.2 New Applications Enabled by CK

We introduce two new applications enabled by CK, i.e., through use of PoCKs. We show how CK enables users to prove facts about asset ownership that they could not easily prove without CK. Specifically, *these applications cannot be securely realized with standard PoKs*.

Key-coupling: In some systems, it is valuable to be able to ascertain that two different private keys, sk_1 and sk_2 (with respective public keys pk_1 and pk_2), are known simultaneously by the same user. This is possible with a CK witness consisting of two private keys. That is, the prover furnishes a CK proof of knowledge of $\mathsf{sk}_1 \parallel \mathsf{sk}_2$, i.e., a concatenation of the two private keys. We refer to such proof as *key-coupling*.

We emphasize that key-coupling is not possible using, e.g., a conventional

[5] The basic idea is to allow users to switch their registered keys secretly; this prevents an adversary from knowing whether a key presented by a user is valid or not. The scheme allows for a race condition between valid users and adversaries with whom keys are shared. It thus does not strictly meet formal definitions of coercion-resistance, e.g., JCJ [154], although use of deposits acts as a practical disincentive to key sharing.

PoK of $\mathsf{sk}_1 \parallel \mathsf{sk}_2$. Two distinct holders of sk_1 and sk_2 could jointly generate such a proof using secure function evaluation and avoid mutual key disclosure.

Key-coupling has a number of applications, particularly in blockchain systems. They include:

1. *KYC dilligence:* Cryptocurrency users often transfer control of their own coins from one key to another, e.g., from hot to cold wallets or vice versa. Users often must undergo know-your-customer (KYC) to transact with exchanges. It can be helpful for a user who has undergone KYC diligence with respect to the address associated with pk_1 to be able to transfer assets to an address associated with pk_2 *without having to undergo diligence again*. By proving simultaneous knowledge of sk_1 and sk_2, the user provides strong evidence that assets transferred between pk_1 and pk_2 belong to the same user—or at least fall under the control of a single entity. The same approach can be used by the owner of the address associated with pk_2 to prove that *she doesn't owe tax on funds* sent from pk_1, as the funds didn't change hands.

2. *Privacy-preserving credential linkage:* Suppose that a user who controls private keys for pk_1 and pk_2 has a public credential attached to pk_2 (e.g., proof of KYC diligence, as above). The user can construct a CK proof for sk_1 *plus* possession of some public key / address (pk_2) with an associated KYC credential. She can do this *without revealing* pk_2.

3. *Enforcing NFT royalty payments:* Non-fungible tokens (NFTs) are blockchain objects that often represent ownership of digital artistic works. Some NFT platforms enforce *royalty payments* to artists (e.g., 5% of sale price) upon resale of an NFT. However, those platforms also support direct, royalty-free transfer between addresses so as to support transfer between addresses belonging to

a single user. As there is no way (prior to our work) to determine single-owner possession of two distinct addresses, users can exploit this royalty-free transfer feature to *bypass* royalty payments. Controversially, for example, a popular marketplace called Sudoswap facilitates royalty-free NFT sales [119]. Key-coupling can, however, enforce *true single-owner possession of addresses in royalty-free transfers*, thereby closing the loophole that deprives NFT creators of ongoing royalty payments.

Note that the ability of key-coupling to distinguish between within-owner and between-owner transfers may not be future-proof: User wallets could eventually support key changes, whether for key rotation or social recovery [46, 51]. This feature could be abused to transfer control between two distinct users while maintaining an appearance of consistent ownership by a single user. Still, the friction against cheating created by key-coupling might still be sufficient to protect, e.g., small-royalty NFT transfers.

CK addresses and Atomic NFTs: Blockchains enable new mechanisms for joint ownership of indivisible digital assets. Such ownership regimes are referred to as *fractionalization*. This is a particularly popular approach to distributing ownership of expensive NFTs among a collection of users.

Fractionalization, however, introduces problems such as price volatility and attractiveness to scammers [239]. As NFTs are essentially financial instruments—and usable not just for digital art, but for real-world assets such as real estate—preventing of fractionalization can also help with know-your-customer (KYC) / anti-money-laundering (AML) compliance, as it ensures that on-chain ownership representation is accurate [247]. Finally, certain types of

NFTs, e.g., *soulbound tokens* (SBTs) [250] are intended by design for exclusive ownership; fractionalization would undermine their utility.

Fractionalization may be prevented on chain, i.e., on the blockchain itself, by allowing ownership only from a user address and not a smart contract. But there exists no mechanism (prior to our work) to prevent *off-chain* fractionalization of NFTs by means of secret sharing or TEEs.

CK offers a novel way to prevent fractionalization of any kind through a concept we call *CK addresses*. A CK address is one whose secret key is guaranteeed to be unencumbered through the use of a PoCK protocol. Through CK addresses, we can create what we refer to as *Atomic NFTs*—NFTs which are designed to only permit ownership by CK addresses. This ensures that at all times *only one entity controls the NFT*.

9.3 Preliminaries and Background

We start by introducing some basic formalism. We introduce some notation and background in this section, including the basic formalism for interactive proof systems [137, 210].

9.3.1 Interactive Proof Systems

Computational model for interactive proofs. We adopt the standard Interactive Turing Machine (ITM) model [137] for protocol execution. An interactive proof system is a pair $(\mathcal{P}, \mathcal{V})$ of ITMs that communicate with each other in

rounds. $\mathcal{P}$ and $\mathcal{V}$ may be given auxiliary inputs z_1 and z_2 respectively, along with a common input x. $\mathcal{V}$ outputs a single-bit at the end of the execution. We use $\langle \mathcal{P}(x, z_1), \mathcal{V}(x, z_2) \rangle$ to denote the random variable for $\mathcal{V}$'s output and $\text{VIEW}_{\mathcal{V}}(\mathcal{P}(x, z_1), \mathcal{V}(x, z_2))$ to denote the random variable for $\mathcal{V}$'s view of the execution.

Interactive proofs of knowledge. Consider a language $L \in \text{NP}$ with witness relation R_L, i.e., $x \in L$ iff. $(x, w) \in R_L$ for a *witness* w. Informally, the goal of a proof of knowledge system is to have the verifier output 1 iff $x \in L$ and the prover "knows" some witness w for x. We say that $(\mathcal{P}, \mathcal{V})$ is an interactive PoK system (for R_L) if it is *complete* and a *proof of knowledge*.

1. *Completeness* means that the honest prover $\mathcal{P}$ can always convince the honest verifier $\mathcal{V}$ when $(x, w) \in R_L$. Concretely, for all $(x, w) \in R_L$, we have

$$\Pr[\langle \mathcal{P}(x, w), \mathcal{V}(x) \rangle = 1] > 1 - \text{negl}(\lambda)$$

where λ is the security parameter.

2. A *proof of knowledge* is a protocol in which, if a (malicious) prover $\mathcal{P}^*$ convinces $\mathcal{V}$ that $x \in \mathcal{L}$ (i.e., $\mathcal{V}$ outputs 1), then the prover "knows" a witness w for x. This is formalized by requiring the existence of an extractor $\mathcal{E}$ that can extract the witness given the description of $\mathcal{P}^*$. More formally, we require that for all $\mathcal{P}^*$ and x,

$$\Pr[w \leftarrow \mathcal{E}^{\mathcal{P}^*}(x) : (x, w) \in R_L] + \text{negl}(\lambda) > \Pr[\langle \mathcal{P}^*(x), \mathcal{V}(x) \rangle = 1].$$

If this property holds only for a computationally bounded (PPT) adversarial prover, $(\mathcal{P}, \mathcal{V})$ is called an interactive *argument* of knowledge system.

3. We further say that a proof of knowledge is *zero-knowledge* if, informally, the verifier learns nothing from a proof execution. Specifically, for any PPT verifier $\mathcal{V}'$, there exists a PPT machine $\mathcal{S}$ (called the *simulator*) such that for all $(x, w) \in R_L$, it holds that

$$\text{VIEW}_{\mathcal{V}'}(\langle \mathcal{P}(x, w), \mathcal{V}'(x) \rangle) \approx \mathcal{S}(x)$$

where $\approx$ means computational indistinguishability.

For a more detailed introduction, we refer the reader to [210].

As stated before, in a PoK system, there is no guarantee that the prover actually has unencumbered access to the witness.

Σ-protocols. A popular type of zero-knowledge proofs of knowledge (ZKPoK) used in practice are Σ-protocols. They are a key building block in our ASIC-based construction. A Σ-protocol is a three-move, interactive ZKPoK with the following structure:

1. $\mathcal{P}$ sends a message a (often called a *commitment*) to $\mathcal{V}$.
2. $\mathcal{V}$ sends a challenge $c \leftarrow\!\!\$\ \{0,1\}^\ell$ to $\mathcal{P}$.
3. $\mathcal{P}$ sends a response s to $\mathcal{V}$.

For a given x, $\mathcal{V}$ decides whether to accept $\mathcal{P}$'s proof based on the proof transcript, which consists of the triple (a, c, s).

Σ-protocols have a property called *special soundness*, whereby an extractor can efficiently compute a witness w from a pair of accepting transcripts (a, c, s)

and (a, c', s') where $c \neq c'$. (They also have a property called *special honest-verifier zero-knowledge*, which means that a simulator given x and c can generate a transcript (a, c, s) distributed like that in a real execution without access to the witness.)

An additional property of many Σ-protocols is *quasi-soundness*, which means that no efficient prover can produce any two transcripts of the form $(a, c, s), (a, c, s')$, where $s \neq s'$, i.e., two different responses for the same commitment-challenge pair. Our construction applies only to quasi-sound Σ-protocols.[6] We assume such Σ-protocols throughout.

The extractor $\mathcal{E}$ for a proof-of-knowledge protocol may in general *rewind* the prover $\mathcal{P}$. In a Σ-protocol, the special-soundness property gives rise to a simple extractor construction. After the first move in the protocol, i.e., the commitment a by $\mathcal{P}$, $\mathcal{E}$ issues a challenge c, and obtains response s. $\mathcal{E}$ then rewinds $\mathcal{P}$ to the point just after the first move and issues a second challenge c', recovering a second response s'. The two transcripts allow extraction of w.

The best-known Σ-protocol—and a practical choice for proving knowledge of a discrete-log-based public key in ASIC-ZKPoCK—is the Schnorr protocol [226], which proves knowledge of a discrete log. Here, $x = g^w$ (where g is a published generator of some suitable group $\mathbb{G}$), and the goal is to prove knowledge of the exponent w.

Straight-line extraction. The need to rewind the prover $\mathcal{P}$ results in loose security reductions for various signature schemes, e.g., [214] and is typically

[6] Given the ability to generate two transcripts (a, c, s) and (a, c, s') for $s \neq s'$, for instance, which is permissible in strongly-sound Σ-protocols [168], a prover can cheat in our protocol.

incompatible with concurrent protocol composition [58]. These issues with rewinding motivated the exploration of *straight-line extractors* (a.k.a. *online extractors*), which do not require rewinding.

Online extractors may observe calls to a hash function H by $\mathcal{P}$, where H is modeled as a *random oracle* (RO), i.e., responses are independent and uniformly random. Of particular interest in our setting are *non-programming* extractors, which involve a strong model where the extractor cannot program the RO, i.e., determine its responses during extraction.

Fischlin [122] proposed a non-interactive ZKPoK scheme with a straight-line extractor whose techniques we adapt to our ASIC construction. In this construction the extractor has access to RO queries. The key idea is that $\mathcal{P}$ uses Σ-protocol transcripts as inputs to the RO in a proof of work (PoW). The scheme is parameterized in such a way that solving the PoW requires with high probability that $\mathcal{P}$ feeds a pair of valid Σ-protocol transcripts $(a, c, s), (a, c', s')$ with $c \neq c'$ to the RO. By observing these transcripts as RO inputs, an extractor $\mathcal{E}$ can extract the witness w.

To provide more detail, Fischlin's scheme builds on a Σ-protocol involving a prover / verifier pair $(\mathcal{P}_\Sigma, \mathcal{V}_\Sigma)$. Parameters include challenge domain cardinality k, proof-of-work difficulty b, global proof-of-work target S, and number of rounds of execution n. Specifically, $H : \{0,1\}^* \to \{0,1\}^b$, and a valid PoW solution is such that $H(z) = 0^b$ for an input z that includes a valid Σ-transcript (a, c, s). The protocol is as follows:

- Prover $\mathcal{P}$: On input (x, w), $\mathcal{P}$ does the following:
  - Runs the first step of n executions of $\mathcal{P}_\Sigma$ to obtain n commitments $\vec{\mathsf{com}} =$

$(a_1, a_2, \ldots, a_n)$. Let $q_i = ([x, i]; \vec{\mathsf{com}})$.

  - For each given a_i, $\mathcal{P}$ completes an execution of $\mathcal{P}_\Sigma$ on challenges $c_j \in [0, k-1]$. Each execution of $\mathcal{P}_\Sigma$ yields a corresponding response s_j. $\mathcal{P}$ sets $\pi_i = (a_i, c_j, s_j)$ for $H((a_i, c_j, s_j) \parallel q_i) = 0^b$ if one exists; otherwise it sets $\pi_i = (a_i, c_j, s_j)$ for the minimal result, i.e., j for $\mathrm{argmin}_j H((a_i, c_j, s_j) \parallel q_i)$.
  - Sends $\vec{\mathsf{com}}$, x, $\vec{\pi} = \{\pi_i\}_{i=1}^n$ to the verifier.

- Verifier $\mathcal{V}$: On input $(x, \vec{\pi})$, $\mathcal{V}$ checks: (1) For all $i \in [1, n]$ that $\mathcal{V}_\Sigma$ accepts π_i; and (2) $\sum_{i=1}^n H(\pi_i \parallel q_i) < S$.

This construction ensures straight-line extraction as follows.

In order to compute PoW results that $\mathcal{V}$ will accept, $\mathcal{P}$ must with high probability hash *at least two* Σ transcripts of the form (a_i, c_j, s_j) for some a_i. That means that the random oracle H will be called on a pair of inputs that include distinct pairs (a_i, c_j, s_j) and (a_i, c'_j, s'_j). From special-soundness, it follows that w can be extracted.

As will be seen later, while our ASIC ACK protocol draws on the technique of combining a Σ-protocol with a PoW, our setting differs considerably. In straight-line extractors, extraction is a theoretical capability used to prove knowledge of w. In our ASIC ACK, extraction is a practical capability used to show that $\mathcal{P}$ can access w. Additionally, in our ASIC ACK, rather than using a *cryptographic* resource in the form of an RO, $\mathcal{P}$ uses a *computational* resource in the form of an ASIC. The result is a protocol that differs somewhat from Fischlin's in terms of the form of oracle queries involved and the resulting security analysis.

Trusted Execution Environments (TEEs). A TEE runs applications with strong confidentiality and integrity protections. Some TEE platforms can issue a type of statement, known as an *attestation*, to untampered execution of a particular application, along with application outputs.

One popular TEE with attestation capabilities is Intel Software Guard eXtensions (SGX) [10, 188, 187]. Trust in the hardware—and in Intel, which authenticates attestation keys—means that only an SGX platform can generate a valid attestation, i.e., attestations are existentially unforgeable. We make use of formalism for SGX-like TEEs in the universal composability (UC) framework from [209]. TEEs are nearly universal today in mobile devices as well, but without built-in attestation capabilities. Google and Apple, however, generate attestations for devices in their ecosystems [12, 13].

We make use of both SGX (Section 9.5) and mobile-device TEEs (Appendix 9.7.2) in different CK variants.

9.4 Proofs of Complete Knowledge

As we have explained, the standard proof-of-knowledge property does not guarantee that the prover actually has unencumbered access to the witness w. This is because in order to recover w, the (knowledge) extractor gets oracle access to the *entire* prover, including parts that may be controlled by separate entities or even the adversary. The issue is that the prover is modeled as a single machine $\mathcal{P}$ even if in reality it is not owned by a single entity. For instance, if w is secret-shared between two independent parties, $\mathcal{P}$ will correspond not to the individual machines, but to the combined system with both parties, and the ex-

tractor is given access to this system. Here, even though the system as a whole knows the witness and the extractor is able to recover it, intuitively, it is clear the neither party alone really has unencumbered access to the witness.

In this section, we formalize *proofs of complete knowledge* (PoCKs), which ensure that a single party has full access to the witness.

9.4.1 Building Intuition

The basic idea behind PoCKs is to design protocols where the knowledge extractor $\mathcal{E}$ is actually runnable in practice, rather than simply a proof construct. We therefore begin with a careful analysis of what can make an extractor fail if practically run, and then use these insights to guide our PoCK formalism. As a first point, since we want $\mathcal{E}$ to be able to run in practice, it is obvious that $\mathcal{E}$ should not rewind $\mathcal{P}$, i.e., it must be *straight-line*. We will further restrict our attention to straight-line (i.e., non-rewinding) extractors that are also *non-programmable* (i.e., unable to program e.g., the random oracle (RO); this is important since real hash functions used to instantiate the RO cannot be programmed). This style of knowledge extractor has been used previously in e.g., [202, 122, 168].

Recall that for standard PoK extraction in the random oracle model, the extractor $\mathcal{E}$ is given two quantities: (1) A transcript of the interaction between the prover and the verifier; and (2) A list of queries (and corresponding responses) made by the prover to the random oracle. What does it mean exactly for $\mathcal{E}$ to be given these inputs in practice? Unfortunately, we find that both of these inputs are problematic to assume in practice (due to encumbrance by MPC or a TEE) which makes designing protocols for our setting particularly challenging.

At an initial glance, the first input seems obtainable—any entity present on the communication channel between the prover and the verifier can observe this transcript (even if the prover is composed of multiple entities and only one communicates with the verifier). This however implicitly assumes that at some point, the communication is not encrypted and can therefore be observed by $\mathcal{E}$. Such an assumption fails if $\mathcal{V}$ is also run inside a TEE, in which case, a prover TEE holding the witness w would be able to convince $\mathcal{V}$ without w being known in plaintext to any non-TEE entity. This observation has a surprising consequence: the trivial PoK of simply sending w to $\mathcal{V}$ *cannot* be a PoCK unless it can be enforced that $\mathcal{V}$ is not run in a TEE (see Remark 12).

The second input—the list of random oracle queries—is also challenging to enforce in light of TEE or MPC encumbrance. This is because a hash function, which will typically be used to instantiate the random oracle, can be computed easily in trusted hardware or MPC. In turn, providing the oracle queries to $\mathcal{E}$ in practice would effectively translate to breaking the trusted hardware or unravelling the MPC protocol to figure out the hash function inputs. Intuitively, the gap here arises from the fact the *physical* instantiation of the random oracle may not comply with the way the extractor functions in theory for the proof of knowledge to go through.

Key technique. To surmount these challenges, we must ensure that the extractor always obtains the inputs required for knowledge extraction in practice. To do so, we consider a physical *resource oracle* functionality $\mathcal{R}$ such that the prover's interaction with $\mathcal{R}$ leaks a witness to $\mathcal{E}$. We seek to deploy $\mathcal{R}$ so that no prover can perform a successful CK proof without $\mathcal{R}$; in a sense, the resource abstraction separates out the part of the protocol responsible for the proof of

knowledge.

Concretely, we model $\mathcal{E}$ as a man-in-the-middle entity for $\mathcal{R}$ that can snoop on the queries made to $\mathcal{R}$. $\mathcal{E}$ will use these queries to extract the witness, thereby giving it (and whoever can see its output) *complete knowledge* of the witness (see Remark 11). To emphasize its physical presence and its non-rewinding nature, we refer to $\mathcal{E}$ as the *eavesdropping extractor*, or simply the eavesdropper.

Looking ahead, in our protocols, $\mathcal{R}$ models special hardware available to $\mathcal{P}$—either a global SGX functionality that attests to seeing the witness, or an untrusted ASIC whose computational speed is superior to that of a trusted environment that can potentially conceal the witness from $\mathcal{E}$. In a sense, the SGX instance represents a *physical manifestation* of a trusted third party that furnishes the witness w directly to $\mathcal{E}$; the ASIC similarly may be thought of as a *physical manifestation* of an RO accessible by $\mathcal{E}$.

Here, $\mathcal{E}$ can be thought of as the machine where $\mathcal{R}$ physically resides; as an example, for the SGX resource, $\mathcal{E}$ represents the host machine of the SGX which may simply be one of the entities within $\mathcal{P}$ or even a different external machine. A crucial point here is that by design, we use a resource $\mathcal{R}$ such that there is no practical resource $\mathcal{R}'$ which provides an identical functionality and allows for encrypted queries to be made directly to $\mathcal{R}'$: this enables $\mathcal{E}$ to view the plaintext queries made to the resource.

Remark 11 (Complete knowledge for *some* entity)**.** Notice that PoCK only guarantees that the eavesdropper $\mathcal{E}$ can extract w. If for some reason, the prover does not have access to the output of $\mathcal{E}$ in practice, then it may not be able to recover w.

As a consequence, PoCK protocols can only guarantee that *some* entity (specifically $\mathcal{E}$ and anyone who can see its output) has complete knowledge of w. For instance, if $\mathcal{R}$ is connected to a different outsourced machine $\mathcal{M}$ instead of $\mathcal{P}$, then $\mathcal{E}$ will correspond to $\mathcal{M}$ and be able to recover w while $\mathcal{P}$ might not. We note that this subtlety is not accounted for in standard PoK formalism since $\mathcal{E}$ does not represent a physical entity.

This property, however, is sufficient to deter the collusion and bribery attacks which motivate our work. For instance, a user will not willingly accept a bribe for her vote if it reveals her key (which also controls her money) to another entity.

9.4.2 Formal PoCK Security

We now formally define proofs-of-complete knowledge. We use λ throughout to denote the security parameter.

Basic setting. Similar to standard PoKs, we consider a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ (modeled as ITMs). Specific to our setting, we model a *resource oracle* $\mathcal{R}$ that can be queried by $\mathcal{P}$. We will often work in a *timed* setting where $\mathcal{P}$ must complete its proof within some time $T(\lambda)$. To model the concrete computational speed of a resource $\mathcal{R}$, we associate with it a function $t_{\mathcal{R}}(\cdot)$ that defines the time taken by the resource to compute responses to its queries. Note that this execution by $\mathcal{R}$ can be concretely faster than by $\mathcal{P}$.

Resource formalism. Abstractly, a resource $\mathcal{R}$ is a randomized and stateful functionality $\mathcal{F}_{\mathcal{R}}$. $\mathcal{R}$ is initialized with an internal state $\mathsf{st}_{\mathsf{initial}} \leftarrow\!\!\$\ \mathcal{R}.\mathsf{Setup}(1^\lambda)$. Upon input inp from $\mathcal{P}$, $\mathcal{R}$ computes $\mathcal{F}_{\mathcal{R}}(\mathsf{st}, \mathsf{inp}) \rightarrow (\mathsf{st}', \mathsf{out})$ where st is the current state of $\mathcal{R}$, st' is the state after the computation, and out is the output returned to prover. We also model the time taken by $\mathcal{R}$ as the randomized function $t_{\mathcal{R}}(\mathsf{st}, \mathsf{inp})$. Note that $t_{\mathcal{R}}(\cdot)$ may be smaller than if the computation was done by $\mathcal{P}$ itself. The tuple $(\mathcal{R}.\mathsf{Setup}, \mathcal{F}_{\mathcal{R}}, t_{\mathcal{R}})$ represents $\mathcal{R}$.

We parameterize our PoCK protocols by a set $\mathfrak{R}$ of "honest" resources, i.e., resources that an honest prover can utilize to convince an honest verifier. We use $\mathfrak{R}_{\mathsf{all}} \supseteq \mathfrak{R}$ to denote the global set of all *practically instantiable* resources—only resources in this set are *assumed* to exist in practice. This explicitly enables modeling our assumption regarding resource practicality (e.g., the non-existence of SGX inside SGX).

PoCK formalism. A T-timed PoCK (when T is unspecified, there are no additional timing constraints) for language $L \in \mathrm{NP}$ with witness relation R_L, a set $\mathfrak{R}$ of honest resources, and a set $\mathfrak{R}_{\mathsf{all}}$ of practical resources is a tuple $(\mathsf{Setup}, \mathcal{P}, \mathcal{V})$ where:

- $\mathsf{Setup}(1^\lambda) \rightarrow pp$ is the setup algorithm that generates public parameters.
- $(\mathcal{P}, \mathcal{V})$ is an interactive proof system where $\mathcal{P}$ is given (pp, x, w) and $\mathcal{V}$ is given (pp, x). $\mathcal{V}$ outputs a single bit indicating whether the prover has complete knowledge of a valid witness for x. For non-interactive proofs, $\mathcal{P}$ will output a proof π which will be given to $\mathcal{V}$ to verify. For T-timed protocols, $\mathcal{P}$ will be required to run in time T.

We can also consider the standard relaxation of *arguments* (instead of proofs) of knowledge for which only PPT provers are considered (although *argument* and *proof* are often used interchangeably in the literature). Our concrete protocols will be "Arguments of Complete Knowledge" (ACK).

Now, for PoCK security, we define in the subsequent paragraphs, two properties—completeness, and forced-revelation (or CK-soundness) that are required to hold.

Completeness. The first PoCK property of *completeness* mirrors the analogous property for PoKs. Recall that completeness states that an honest prover who holds the witness can convince the verifier to output the success bit. The only difference now for PoCK completeness is that the prover is endowed with a resource oracle $\mathcal{R}$. Formally, PoCK completeness states that for all pp, $\mathcal{R} \in \mathfrak{R}$, and $(x, w) \in R_L$:

$$\Pr[\langle \mathcal{P}^{\mathcal{R}}(pp, x, w), \mathcal{V}(pp, x)\rangle = 1] > 1 - \mathrm{negl}(\lambda).$$

Furthermore, $\mathcal{P}^{\mathcal{R}}$ runs in time at most $T(\lambda)$.

Forced revelation or CK-soundness. The second PoCK property of *forced revelation* is similar in spirit to the knowledge-soundness property of standard PoKs. Abstractly, if a prover is able to convince an honest verifier using any practical resource (i.e., within $\mathfrak{R}_{\mathsf{all}}$ and not just within $\mathfrak{R}$), then the eavesdropper will be able to output a valid witness. Let $\mathtt{out}_{\mathcal{E}}$ denote the output of $\mathcal{E}$. A party able to view the output of the eavesdropper obtains the full witness. Formally, for all

pp, inputs x, provers $\mathcal{A}$, and resources $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$ such that $\mathcal{A}^{\mathcal{R}'}$ runs in time $T(\lambda)$,

$$\begin{aligned}&\Pr[\langle \mathcal{A}^{\mathcal{R}'}(pp, x), \mathcal{V}(pp, x)\rangle = 1] \\ &\qquad < \Pr[x \in L \ \wedge \ (x, w = \mathtt{out}_{\mathcal{E}}) \in R_L] + \mathsf{negl}(\lambda).\end{aligned}$$

Forced revelation directly implies a couple of nice properties. First, it implies the usual soundness notion since no prover can convince $\mathcal{V}$ of an $x \notin L$ (except with negligible probability). Second, it also implies that if the prover does not make use of any resource, making it so that $\mathcal{E}$ cannot eavesdrop, then it cannot convince $\mathcal{V}$ except with negligible probability *even if it has the witness*. Intuitively, this property is necessary because otherwise it would imply the ability to prove CK through e.g., a 2PC protocol where the witness is encumbered. As illustrated in the remark that follows, forced revelation also has surprising ramifications, which underscore the nuances of working in our PoCK setting.

Forced revelation implies an interesting separation between trivial protocols for PoK and PoCK, detailed in the following remark.

Remark 12 (Trivial PoK and PoCK protocols)**.** Forced revelation implies an interesting separation between trivial protocols for PoK and PoCK. Recall that a trivial PoK protocol is for $\mathcal{P}$ to simply provide the witness to $\mathcal{V}$. Of course, such a protocol does not offer any privacy (e.g., zero-knowledge) properties. Notice however, that this would not be a PoCK protocol (since $\mathcal{E}$ does not come into play). This may seem surprising but is in fact an important consequence of the PoCK setting.

Abstractly, if the verifier receives the witness through an encrypted channel (e.g., through TLS), then it would also be possible for two parties constituting the prover and holding only shares of w to directly compute the required en-

cryption of w for the channel. In such a case, the trivial PoK protocol could be simulated by two prover parties, neither of which has complete knowledge of w. Importantly, this is also possible if $\mathcal{V}$ is run inside a TEE since there is no point at which w is revealed in plaintext. Consequently, simply sending w will not be a PoCK protocol.

Intuitively, if we can guarantee that $\mathcal{V}$ is not run inside a TEE, then it will be possible to eavesdrop on the witness, making the trivial protocol a PoCK. One way this can be done is by having $\mathcal{V}$ itself be a TEE instance with the assumption that it is not practical to run a TEE inside another TEE.

This unearths a dependence of PoCKs on how the communication to $\mathcal{V}$ is defined, which is not seen for standard PoKs.

9.4.3 Zero-Knowledge PoCK

Most applications require the prover's witness to not be leaked to the verifier. A strong property often considered in the PoK realm is that of *zero-knowledge* (ZK) [137]. Informally, this ensures that *no additional information* is leaked to the verifier. We will add a similar requirement to PoCKs to formalize "Zero-Knowledge Proofs of Complete Knowledge" or ZKPoCKs.

Zero-Knowledge property for PoCKs. We now adapt the standard zero-knowledge definition to our setting. Formally, we define the zero-knowledge property as follows: For any PPT verifier $\mathcal{V}'$, there exists a PPT machine $\mathcal{S}$ (called the simulator) such that for all $\mathcal{R} \in \mathfrak{R}$, $(x, w) \in R_L$ and auxiliary input

$z \in \{0,1\}^*$, it holds that:

$$\text{VIEW}_{\mathcal{V}'}(\mathcal{P}^{\mathcal{R}}(pp, x, w), \mathcal{V}'(pp, x, z)) \approx \mathcal{S}^{\mathcal{R}}(pp, x, z).$$

9.4.4 Eavesdropper Undetectability

The PoCK formalism models $\mathcal{E}$ as a concrete man-in-the-middle entity which eavesdrops on queries made to $\mathcal{R}$. We implicitly assume that $\mathcal{E}$ is *always run*; in other words, we do not model a scenario where w was not extracted even though it *could have been*. While this distinction is not important in the standard cryptographic context, and therefore not part of our core PoCK formalism, it uncovers subtleties in the context of side channels and incentive compatibility. We formalize this as the property of "eavesdopper undectability" and remark on its utility below.

Side channel on the usage of $\mathcal{E}$. To differentiate between whether $\mathcal{E}$ was used to recover the witness or not, we consider a side channel that provides a remote adversary $\mathcal{A}$ with this information (explicitly, this is added to the VIEW of an adversarial verifier). Consequently, $\mathcal{A}$ can now take actions based on whether the witness was extracted; notably, as we show later, this gives $\mathcal{A}$ additional advantages in our vote bribery scenario.

It is important to emphasize here that the witness can always be extracted, (thereby satisfying CK-soundness); the difference is in *whether it actually was*. We also note that this detection ability is a highly unconventional power given to the adversary; while $\mathcal{A}$ cannot itself recover the witness, it is still made aware of whether $\mathcal{P}$ did. This is non-standard in the context of existing literature—when

$\mathcal{R}$ (and $\mathcal{E}$) is not in the domain of $\mathcal{A}$ (because otherwise $\mathcal{A}$ could extract the witness itself), we note that there is likely no practical side channel that reveals to $\mathcal{A}$ whether $\mathcal{P}$ chose to extract.

Still, to ensure that PoCK protocols can be correctly deployed in applications where side channels need to be accounted for, we introduce an explicit assumption—*eavesdropper undetectability*—as the property that an adversary cannot detect whether $\mathcal{E}$ was run or not. We show how this property is critical for incentive-compatibility in our vote bribery example. This also serves to highlight the non-triviality of our CK setting.

Formal description. To define eavesdopper undectability, we relax our earlier modeling assumption that $\mathcal{E}$ exists as a man-in-the-middle entity for $\mathcal{R}$ and can snoop on any queries made to it. Instead, we will give the prover the ability to make queries to $\mathcal{R}$ *without the usage of* $\mathcal{E}$; we use $\mathcal{R} \setminus \mathcal{E}$ to denote this oracle. The honest prover $\mathcal{P}$ will still make use of $\mathcal{E}$.

We can now define eavesdopper undectability as the following property: For pp, inputs x, and $\mathcal{R} \in \mathfrak{R}$, for all (possibly malicious) provers $\mathcal{P}'$ and verifiers $\mathcal{V}'$, there exists $\widehat{\mathcal{P}}$ such that the following ensembles are indistinguishable:

$$\begin{aligned} &\text{VIEW}_{\mathcal{V}'}\left(\mathcal{P}'^{\mathcal{R}\setminus\mathcal{E}}(pp,x),\mathcal{V}'(pp,x)\right) \\ \approx\ &\text{VIEW}_{\mathcal{V}'}\left(\widehat{\mathcal{P}}^{\mathcal{R}}(pp,x),\mathcal{V}'(pp,x)\right). \end{aligned}$$

Intuitively, this means that no $\mathcal{V}'$ can distinguish whether it is interacting with a prover which uses $\mathcal{E}$ or one which does not.

In the example that follows, we briefly describe how a side channel which informs $\mathcal{A}$ of whether $\mathcal{E}$ was run breaks security in our vote bribery scenario.

Eavesdropper undetectability is required here to prevent this attack. We leave further exploration of this property to future work.

Incentive compatibility example. Suppose that a PoCK Π is used in a voting application to mitigate the risk of key-encumbrance based bribery. As stated earlier, Π ensures that the eavesdropper $\mathcal{E}$ *can recover* the secret key but says nothing about *whether the recovery is actually carried out*—our previous CK formalism implicitly assumes that $\mathcal{E}$ will always output the recovered key.

Still, if there was some side channel through which a remote adversary $\mathcal{A}$ could detect whether the key was extracted (by $\mathcal{E}$ or $\mathcal{P}$ in general), then it could condition its bribe on this extraction action not being taken. The consequence of such a conditional bribe is that while $\mathcal{P}$ has the ability to learn her key, she will be *incentivized not to do so* in order to profit from the bribe. In particular, while $\mathcal{P}$ *can* always learn her full key, making the protocol satisfy CK-soundness, $\mathcal{A}$ will be able to detect such an action and refuse to pay $\mathcal{P}$; this incentivizes $\mathcal{P}$ to not learn her key even if she is able to. The key will therefore remain encumbered. Eavesdropper undetectability removes this side channel vulnerability.

9.5 SGX-based PoCK Protocol

We now describe SGX-PoCK, a simple but illustrative PoCK protocol which uses an SGX TEE as its resource. Intuitively, for this PoCK, the SGX models a physical manifestation of a trusted third party to whom the prover will submit the witness. We use SGX for concreteness but note that a similar PoCK can be realized through any TEE which admits remote attestation, including those in

mobile devices, as discussed in Section 9.7.2.

SGX resource. We use the formalism for TEEs with attested execution from Pass et al. [209]. Abstractly, SGX attestation is modeled using a global functionality (i.e., with global setup [59]).

$\mathcal{G}_{\mathsf{SGX}}$ models all valid SGX processors and is initialized with a master key pair $(\mathsf{mpk}, \mathsf{msk})$ with signature scheme $\mathrm{S} = (\mathrm{S.kg}, \mathrm{S.sign}, \mathrm{S.ver})$; this intuitively allows the modeling of anonymous attestation which prevents identifying the SGX which signed an attestation.

$\mathcal{G}_{\mathsf{SGX}}$ permits SGX-equipped parties (denoted by the set Reg) to install programs on their SGX and compute outputs. When a party X provides input inp to an installed program prog, $\mathcal{G}_{\mathsf{SGX}}$ computes its output out and a signature σ on $(\mathsf{id}, \mathsf{prog}, \mathsf{out})$ where id denotes any relevant session identification information. The tuple (out, σ) is then sent to X as the attested output. For completeness, we detail the full $\mathcal{G}_{\mathsf{SGX}}$ functionality in Fig. E.1 in Appendix E.1.

SGX-PoCK description. We describe the full SGX-PoCK protocol in Fig. 9.2. The public parameter pp output by Setup is just the SGX public key mpk. For the proof protocol, the prover $\mathcal{P}$ first installs the program $\mathsf{prog}_{\mathsf{CK}}$ through $\mathcal{G}_{\mathsf{SGX}}$. Now, given (x, w) in the relation R_L as input, $\mathcal{P}$ submits the tuple $(\text{“expose”}, x, w)$ to $\mathcal{G}_{\mathsf{SGX}}$ and gets back a signature σ on $(\mathsf{id}, \mathsf{prog}_{\mathsf{CK}}, (\text{“exposed”}, x))$ which it forwards to $\mathcal{V}$. By checking the validity of the signature, $\mathcal{V}$ can convince itself of complete knowledge of a witness corresponding to x.

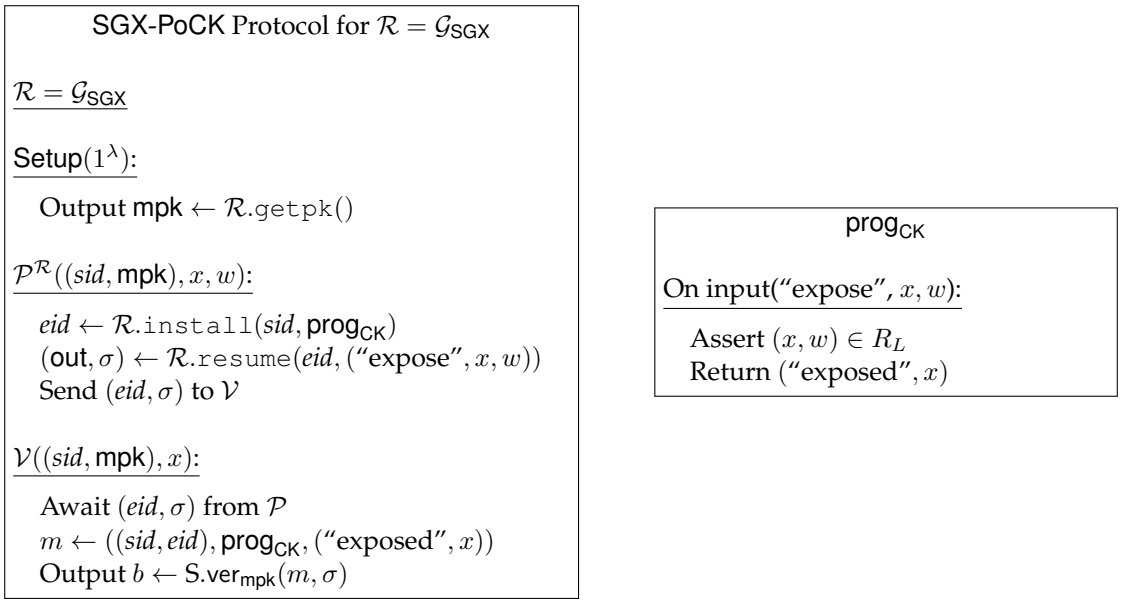

Figure 9.2: SGX-PoCK Protocol Description.

9.5.1 SGX-PoCK Properties

It is easy to see that an honest $\mathcal{P}$ given w can always convince $\mathcal{V}$; in other words, completeness holds for SGX-PoCK. For CK-soundness, we require an assumption on the infeasibility of specific types of resources, as we describe below:

Resource assumptions. To ensure that the witness w exposed to the SGX can be eavesdropped upon, intuitively we need to assume that it is not practical to run $\mathsf{prog}_{\mathsf{CK}}$ in an SGX within another SGX. This is because otherwise, the outer SGX could be in possession of w which it could expose to the inner SGX to obtain a CK proof without w ever being accessible outside of a trusted enclave. We briefly remark on how this assumption can be removed in Appendix E.1.

In the context of the $\mathcal{G}_{\mathsf{SGX}}$ formalism, this means that no program prog installed by a party can install its own program prog′—this is implicitly assumed within [209] since only a fixed registration set Reg is considered for SGX de-

vices.

CK-soundness proof. Now, assuming that there is no practical resource $\mathcal{R}' \subseteq \mathfrak{R}_{\mathsf{all}}$ that models such a 2-layer SGX, it is easy to show that SGX-PoCK satisfies CK-soundness. We provide a sketch below and defer the full game-based proof to Appendix E.1.1.

Suppose that some $\mathcal{P}'$ is able to convince the honest $\mathcal{V}$ that it knows the witness to a statement x. This can happen in only one of two ways: (1) $\mathcal{P}'$ submits (x, w) to the honest resource $\mathcal{R} = \mathcal{G}_{\mathsf{SGX}}$ as one of its queries; (2) $\mathcal{P}'$ does not query $\mathcal{G}_{\mathsf{SGX}}$ with (x, w)—it either does not use $\mathcal{G}_{\mathsf{SGX}}$ at all (potentially uses a different $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$) or queries it with different values. In the first case, w will be sent in plaintext to $\mathcal{G}_{\mathsf{SGX}}$ allowing $\mathcal{E}$ to easily output it (since we assume that $\mathcal{G}_{\mathsf{SGX}}$ cannot be simulated inside another TEE). The second case implies that $\mathcal{P}'$ was able to forge a valid $\mathcal{G}_{\mathsf{SGX}}$ signature on (x, w) using either (i) a different resource without the msk of $\mathcal{G}_{\mathsf{SGX}}$ or (ii) valid signatures from $\mathcal{G}_{\mathsf{SGX}}$ on different $x' \neq x$. Both of these contradict the SUF-CMA security of the signature scheme S used and therefore only arise with negl(λ) probability.

Remark 13 (SGX inside SGX)**.** SGX-PoCK can in fact be modified to work even when an SGX can be run inside another SGX as long as this can be done only a finite number of times. Specifically, if is practical to run k layers of SGX but not $k + 1$ layers, then SGX-PoCK can be modified to use the k-layer SGX as the resource $\mathcal{R}$; $\mathcal{P}$ now obtains an attestation from this $\mathcal{R}$. By assumption, since a $(k + 1)$-layer SGX is not practical, the witness submitted to $\mathcal{R}$ will be seen by $\mathcal{E}$ allowing for extraction.

This also serves to future proof our protocol in case of advances in TEE in-

frastructure.

Privacy properties. Observe that SGX-PoCK as described is not zero-knowledge since the attestation can be forwarded. Still, the SUF-CMA security of S implies non-trivial privacy properties over the basic PoCK. In particular, given many SGX-PoCK proofs (which are just signatures under the master key), an adversary still cannot forge a different SGX-PoCK proof for another statement x.

Making SGX-PoCK satisfy zero-knowledge. Intuitively, to make the protocol zero-knowledge, there must exist a simulator $\mathcal{S}$ that can simulate the protocol transcript without access to the witness. Further note that $\mathcal{S}$ should also not be able to program the master secret key of the SGX (this is accounted for since we model the SGX resource as a GUC functionality).

We now briefly describe how SGX-PoCK can be made zero-knowledge. As is the case with other GUC protocols, $\mathcal{S}$ will require a trapdoor in other to simulate transcripts. Towards this, intuitively, we introduce a trapdoor τ which allows the generation of arbitrary SGX attestations for our protocol.

Specifically, $\mathcal{V}$ first chooses a trapdoor τ and submits it to $\mathcal{G}_{\mathsf{SGX}}$ which returns an attestation σ_c on $c = \mathsf{owf}(\tau)$ where owf is a one-way function. $\mathcal{V}$ sends the (c, σ_c) to $\mathcal{P}$ who then submits (c, x, w) to $\mathcal{G}_{\mathsf{SGX}}$. Before providing an attestation that the witness w was seen, $\mathcal{G}_{\mathsf{SGX}}$ ensures that the correct c was input. Finally, the $\mathcal{V}$ can check the correctness of the attestation to complete the proof. Note that this protocol also requires $\mathcal{V}$ to possess an SGX device.

This construction will now be zero-knowledge since the simulator $\mathcal{S}$ can use

the trapdoor τ to forge attestations and simulate the interaction between $\mathcal{P}$ and $\mathcal{V}$. The proof is analagous to the one from [209, Thm. 19].

9.6 ASIC-Based PoCK Construction

In this section, we explore the design of a (ZK)PoCK using a cryptocurrency mining ASIC as the prover resource $\mathcal{R}$—a protocol we call ASIC-ZKPoCK. Our construction is quite general. It can transform a broad class of Σ-protocols [226]—a common class of three-move, honest-verifier ZKPoK—into an (honest-verifier) ZKPoCK through the use of an ASIC. The only requirement is that the Σ-protocol be quasi-sound.

Intuition. ASIC-ZKPoCK makes use of the *performance gap* between computation in secure environments (e.g., SGX) or secure multi-party computation (MPC) and computation using fast ASIC hardware. By running as a time-constrained protocol, ASIC-ZKPoCK ensures that it is only feasible to compute a correct, timely proof using a mining ASIC.

As required, mining ASIC hardware has an eavesdropping channel $\mathcal{E}$. (Mining ASICs, as we explain below, don't support encryption, so eavesdropping is straightforward.) This channel $\mathcal{E}$ allows the prover to extract the witness during the proof generation process, ensuring complete knowledge.

In short, a mining ASIC may be viewed as a computing resource $\mathcal{R}$ that is special in that it is fast—faster than a CPU—and has an eavesdropping channel $\mathcal{E}$ on inputs. A mining ASIC thus fits our basic CK framework shown in Fig. 9.1.

To additionally show practicality, we show how ASIC-ZKPoCK can be parameterized to work with cheap outmoded ASICs (which are no longer practical for e.g., Bitcoin mining).

Why ASIC-based (ZK)PoCKs? There are two reasons, security and performance related respectively, for exploring ASIC-based PoCKs over TEE-based PoCKs. First, many TEE-based machines operate in the cloud. As noted in Section 9.1, TEE vulnerabilities could expose the private keys to cloud operators or remote adversaries with access to the cloud. Second, in the case of blockchain applications, TEE attestations can be expensive to verify. For example, EPID [150] attestations—an attestation type generated by Intel SGX without special provisioning and with optional privacy protection—is expensive to verify in the Ethereum Virtual Machine (EVM).

9.6.1 Background: Mining ASICs and PoWs

Cryptocurrencies like Bitcoin use *proof of work* (PoW) [152] for the safety of their underlying consensus mechanism for block generation, or *mining*. PoW involves solving puzzles (see below) by means of repeated cryptographic hashing.

A mining ASIC, designed for fast PoW puzzle solving can compute hashes 1,000,000× faster than a CPU—and thus achieves a performance gap compared to any SGX-protected application. Mining ASICs today take only *unencrypted* inputs, meaning that their inputs are exposed to users.[7]

[7]Even if such ASICs were ultimately to support encryption, if they do not also support enclaves, any value input to an ASIC will still be exposed to the user. While one could imagine

A PoW puzzle is based on a particular hash function $\mathcal{H}$ (typically modeled as a random oracle) with ℓ-bit outputs, where ℓ is a security parameter. A puzzle instance has a *difficulty* d corresponding to the probability of correctly solving it with a single hash computation.[8] A puzzle instance may also include ancillary data B (e.g., the block header data for Bitcoin). Solving a puzzle with data B and difficulty $d \in [1, 2^\ell)$ involves finding a *nonce* ν such that $\mathcal{H}(B \parallel \nu) < 2^\ell/d$. The probability of solving the puzzle for any random nonce is an independent and identically distributed Bernoulli random variable with success probability $1/d$ (for $d \mid 2^\ell$).

ASIC-ZKPoCK **intuition.** The idea behind ASIC-ZKPoCK is to require $\mathcal{P}$ to use an ASIC to find a solution π to a PoW puzzle. π is required to include a proof of knowledge of the witness w. Specifically, $\mathcal{P}$ specifies PoK commitment a and π includes the challenge c and response s for a Σ-protocol transcript (a, c, s) involving w. Since Σ-protocols are zero-knowledge, the transcript (and thus π) will not expose w.

Despite this, the process of *computing* π will involve $\mathcal{P}$ sending *multiple* Σ-protocol transcripts to the ASIC. Consequently, given the special soundness (and quasi-soundness) of the Σ-protocols we use, the eavesdropper $\mathcal{E}$ will be able to *extract* w *with high probability*.

While it is possible in principle to perform mining for an ASIC-ZKPoCK in an SGX enclave (in a CPU), this is not realistic, as ASICs are far more performant than CPUs. As an example, a top-of-the-line ASIC can outperform a (single-

reasons to support encryption in mining ASICs, there's no compelling reason to support enclaves.

[8] We refer to d generically in our protocol description as a difficulty parameter, without reference to the specific notion of "difficulty" in Bitcoin.

server) SGX application by a factor of more than 1,000,000.

9.6.2 ASIC-Based PoCK: Protocol construction

Formally, in ASIC-ZKPoCK, $\mathcal{P}$ and $\mathcal{V}$ execute a Σ-protocol. $\mathcal{P}$ embeds a valid proof transcript (a, c, s) for the Σ-protocol in a PoW puzzle whose solution π constitutes a full ASIC-ZKPoCK proof. The key idea in our construction is to *require $\mathcal{P}$ to try out multiple puzzles, each with a different challenge c (and thus response s), in order to find a solution*. We accomplish this by carefully choosing parameters such that with high (but still constant) probability, a single randomly chosen c will not lead to a puzzle with a valid solution. As a result, $\mathcal{P}$ must *input different transcripts to the ASIC* (or in other words, create different puzzles for the ASIC), among which, by quasi-soundness, is a pair $(a, c, s), (a, c', s')$ with $c \neq c'$ and $s \neq s'$. From this pair, given the special soundness property of the Σ-protocol, $\mathcal{E}$ can extract w.

At the same time, however, π itself—the solution revealed to $\mathcal{V}$—contains *only one proof transcript*. Thus $\mathcal{V}$ does not learn w ensuring that the protocol remains zero-knowledge.

As $\mathcal{P}$ must complete the proof in a limited period, it can succeed only with a powerful resource. Given the right parameter choices, this means that $\mathcal{P}$ must employ an ASIC. We will consider the set $\mathfrak{R}$ of (honest) resources that work to be all PoW ASICs with hash rate at least some threshold Q_{asic}.

Preliminaries. ASIC-ZKPoCK involves a random PoCK *challenge* r from $\mathcal{V}$. Let $\pi = (B, \nu)$ denote a puzzle solution computed by $\mathcal{P}$ in response to a PoCK

challenge r. The puzzle solution π consists of a block header B and nonce ν corresponding to a valid proof of work. Let $\Sigma\mathsf{map}_r(\pi) \to (c, s)$ denote a function, dependent on r (as specified below), that maps π to a pair (c, s). We define two verification functions:

- $\mathsf{PoKAccept}(x, (a, c, s)) \to \{\mathsf{true}, \mathsf{false}\}$ checks the correctness of the Σ-protocol transcript (a, c, s) w.r.t. public PoK value x.
- $\mathsf{puzAccept}[d, \beta](\pi) \to \{\mathsf{true}, \mathsf{false}\}$ checks that (B, ν) represents a correct puzzle solution with difficulty d, i.e., $\mathcal{H}(B, \nu) < 2^\ell/d$. puzAccept also checks that nonce ν is of correct size ($\nu < \beta$), for a parameter β discussed below.

Our ASIC-ZKPoCK protocol involves n rounds of the above form where a challenge r is sent by the verifier and a solution π must be computed by the prover in the round time τ. For the verifier to accept the proof, the prover must compute the puzzle solution in more than y rounds. Let $\mathsf{puzAccept}^y[d, \beta](\pi_1, \ldots, \pi_n)$ denote whether more than y (out of n) puzzle solutions are valid. Looking ahead, setting y appropriately will ensure that the proof can be completed by an honest ASIC but not by an adversarial prover.

Efficient puzzle-solving vs. transcript extraction. As explained above, our protocol design must force $\mathcal{P}$ to try different values of c while computing a puzzle solution π. Changing c, however, carries the overhead of computing a new corresponding block header B and feeding B to the ASIC. While this is not relevant for a "theoretical ASIC," it poses a significant practical challenge. Therefore, to obtain a protocol which can be deployed, we don't want $\mathcal{P}$ to have

to change c *too* frequently. This requires us to correctly set the nonce size (in particular $\nu < \beta$) as discussed below.

Our approach to resolving this tension is to: (1) map B to a distinct (c, s), so that changing B changes (c, s) *but* (2) allow $\mathcal{P}$ to explore a range of different nonces ν for a given B.

We construct a mapping $\Sigma\mathsf{map}_r(\pi) \to (c, s)$ as follows. Recall that $\pi = (B, \nu)$. The function $\Sigma\mathsf{map}_r$ partitions $B \to B[1] \parallel B[2]$. It computes $c = \mathcal{H}(B[1] \parallel r)$ from $B[1]$. Here, $\mathcal{H}(\cdot \parallel r)$ can be viewed as a random hash function selected by $\mathcal{V}$ (using PoW challenge r) to prevent PoW precomputation.

Because s depends on c (and a), $\Sigma\mathsf{map}_r$ is constructed such that $\mathcal{P}$ can specify $s = B[2]$, i.e., encode s in a portion of B distinct from that for c. Given the collision-resistance of $\mathcal{H}$, changing $B[1]$ of course changes c. Changing $B[2]$ also changes $B[1]$: Given a fixed a, a given c has only one corresponding response s, due to the quasi-soundness property discussed in Section 9.3. In short, $\mathcal{P}$ cannot feasibly construct distinct blocks that map to the same (c, s). At the same time, we allow $\mathcal{P}$ to explore the space of possible nonces ν. To force $\mathcal{P}$ to try multiple puzzles (and thus multiple (c, s)), the nonce space should not be too large. Therefore, we impose in puzAccept the restriction $\nu < \beta$ for a protocol parameter β.

In summary, $\mathcal{P}$ can feed a block header B to an ASIC to solve a puzzle corresponding to some challenge c. Provided that the security parameter β is small enough—i.e., the space of valid ν is small—the probability of $\mathcal{P}$ finding a puzzle solution $\pi = (B, \nu)$ for any single value of c is low. Therefore, $\mathcal{P}$ must w.h.p. try multiple values of c to find a puzzle solution. Consequently, $\mathcal{P}$ is likely to use a

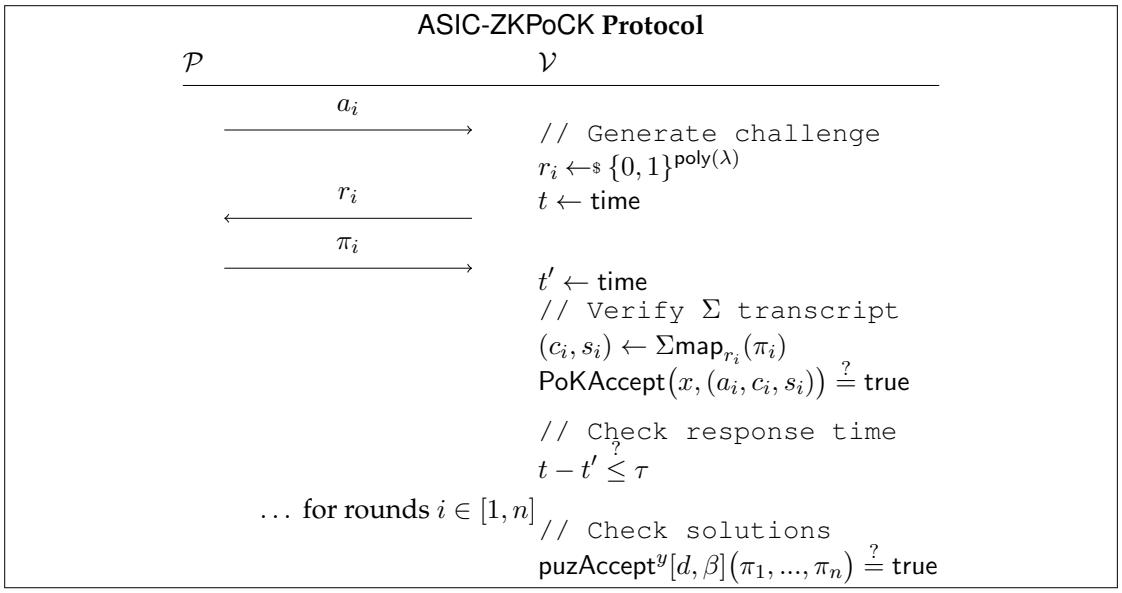

Figure 9.3: ASIC-ZKPoCK protocol. The protocol executes over n rounds. $\mathcal{V}$ runs PoKAccept in each round to check that $\mathcal{P}$ given a valid Σ transcript (a_i, c_i, s_i) and that $\mathcal{P}$ has run within time bound τ. After n rounds, $\mathcal{V}$ runs $\mathsf{puzAccept}^y$ to check more than y out of the n PoW puzzle solutions $\pi_1, ..., \pi_n$ are correct with respect to the difficulty specified by d.

pair of triples $(a, c, s) \neq (a, c', s')$ from which w can be extracted.

ASIC-ZKPoCK protocol. The $\mathsf{Setup}(1^\lambda)$ algorithm simply provides all the concrete parameters (described later) to both parties. The protocol between $\mathcal{P}$ and $\mathcal{V}$ is presented in Fig. 9.3. Two further points are worth highlighting. First, we reiterate that the protocol is *interactive*; $\mathcal{V}$ supplies a PoW challenge r as described above. Second, the protocol is *timed*; $\mathcal{P}$'s response is only accepted by $\mathcal{V}$ if it is returned within time τ. The goal, again, is to ensure that computation has taken place in a (fast) ASIC, rather than a (relatively slow) CPU.

B	Block header
ν	PoW puzzle nonce
π	PoW solution
r	PoCK challenge
(a, c, s)	Σ-protocol transcript

d	PoW puzzle difficulty
y	PoW puzzle threshold
τ	Prover time bound
β	Bound on nonce ν size
$\lambda = \ell$	Security parameter
n	Total number of rounds

Figure 9.4: Protocol notation (left) and parameters (right)

9.6.3 Security Analysis

In this section, we prove security of ASIC-ZKPoCK by showing that it satisfies the PoCK properties of completeness, forced-revelation, and (honest-verifier) ZK. Later, we demonstrate its practicality by choosing concretely viable parameters. The detailed proofs are deferred to the full version. We begin with some simple results.

Fact 9.1. Consider $Y \sim \mathsf{Binomial}(n, p)$ and let $\mathsf{F}(n, p, y) = \Pr[Y \leq y]$. If $y < np$, then $\mathsf{F}(y, n, p) \leq exp(-\frac{(np-y)^2}{3np})$. If $n, y = \Theta(\lambda)$, and $p, y/n = \Theta(1)$ with $y/n < p$, then there exists a negligible function $\mathrm{negl}(\lambda)$ such that $\mathsf{F}(y, n, p) \leq \mathrm{negl}(\lambda)$. Alternatively, if $y/n > p$, then $\mathsf{F}(n - y, n, 1 - p) \leq \mathrm{negl}(\lambda)$.

Proof. This follows directly from the lower-tail Chernoff bound. □

Lemma 9.2. *Let p^Q_{one} denote the probability of $\mathcal{P}$ with hash rate Q successfully computing one PoW puzzle solution in time τ. Define $k = \tau Q/d$. Then: $p^Q_{\mathsf{one}} = 1-(1-1/d)^{kd}$.*

Proof. In time τ, $\mathcal{P}$ can compute $Q\tau = kd$ hash evaluations. Recall that the probability of finding a solution for a single evaluation is $1/d$. Therefore, the probability of finding a solution in kd evaluations i.e., p^Q_{one} can be given by $1 - (1 - 1/d)^{kd}$. □

Setting parameters. Looking ahead, to obtain asymptotic security, we set the parameters as follows: d, β are constants in λ. For simplicity, our analysis considers Q and τ to also be constants. We note however that since λ changes the output of the hash function, it may be the case that the hashrate Q degrades with λ (e.g. $Q = \Theta(1/\lambda)$). Here, the round time τ can be increased to maintain $\tau Q = \Theta(1)$; the analysis remains the same. Further, n, y will be $\Theta(\lambda)$ such that y/n is a constant.

Completeness proof. Consider the PoW threshold satisfying $y < np_{\text{one}}^{Q_{\text{asic}}}$. Now, if n and y are $\Theta(\lambda)$ with $y/n = \Theta(1)$, then by Fact 9.1, the completeness error for the honest prover will be $\text{negl}(\lambda)$, which proves completeness of ASIC-ZKPoCK.

CK-soundness proof. Notice first that for ASIC-ZKPoCK to function correctly, it should be infeasible for an adversary to execute the protocol in an enclave (thus on a CPU). Unfortunately, mining puzzles are embarrassingly parallel, which means that in principle, an adversary can use a network of multiple TEE-enabled hosts to solve them. We must therefore characterize security in terms of the size of the network, say m.

Furthermore, even assuming that an ASIC is used, in order for $\mathcal{E}$ to extract the witness, we also need to show that at least two distinct challenges c and c' are used in the computation.

Concretely, for CK-soundness, we need to bound the adversarial success probability in two strategies: (1) Compute a valid puzzle solution using only one challenge; (2) Compute a valid puzzle solution using m TEE-enabled CPUs. In the full version, we show, through a game-based analysis, why it is sufficient

to only consider these approaches.

We note that while the adversary can send one challenge each to m different ASICs (to prevent extraction caused by two challenges sent to a single ASIC), this requires the adversary to use separate machines for each ASIC and requires it to enforce strong non-collusion properties between these outsourced machines, which is infeasible. We note this in Remark 14 but still allow the adversary to use this strategy within our analysis. Additionally, as noted earlier, we assume, as is the case in practice, that the ASICs do not work over encrypted or secret-shared inputs, avoiding MPC over ASIC strategies[9] (see the proof in the full version for details) [ToDo: fix. point to appendix].

Lemma 9.3. *Let p^{onechal} be the probability (irrespective of hash-rate) of successfully computing a valid puzzle solution using a single challenge c. Then: $p^{\mathsf{onechal}} = 1 - (1 - 1/d)^{\beta}$.*

Proof. The probability of finding a solution for a single evaluation is $1/d$. Since the nonce-space is of size β, the probability that no nonce works is $(1 - 1/d)^{\beta}$. Therefore the probability p^{onechal} that some nonce works is $1 - (1 - 1/d)^{\beta}$. □

Now, let $\mathbb{P}^{y,\mathsf{onechal}}_{\mathsf{succ}}$ denote the probability that this happens in more than y (out of n) rounds, i.e., the adversary wins using strategy (1). Notice that $\mathbb{P}^{y,\mathsf{onechal}}_{\mathsf{succ}} = \mathsf{F}(n - y, n, 1 - p^{\mathsf{onechal}})$, and so if $\beta = \Theta(1)$ is such that $p^{\mathsf{onechal}} < y/n$, then $\mathbb{P}^{y,\mathsf{onechal}}_{\mathsf{succ}}$ is negligible in λ.

Following (2), the adversary seeks to boost its mining rate while preventing disclosure of w by using a network of m enclave-enabled CPUs. Let Q_{cpu} denote

[9] Even if future ASICs do support such MPC, we posit that ASICs which do not support this will always be faster, allowing us to appropriately parameterize our protocol.

the fastest hash rate achievable in an enclave and consider an adversary that uses a network of m enclave-enabled CPUs, i.e., $Q_{\mathsf{adv}} = mQ_{\mathsf{cpu}} \ll Q_{\mathsf{asic}}$. Consider $p_{\mathsf{one}}^{Q_{\mathsf{adv}}} < y/n = \Theta(1)$. Fact 9.1 directly implies that an adversarial prover succeeds in more than y rounds with at most negligible probability.

In essence, ASIC-ZKPoCK achieves both completeness and CK-soundness when we set the success threshold y such that $np_{one}^{mQ_{\mathsf{cpu}}} < y < np_{one}^{Q_{\mathsf{asic}}}$ and β such that $np^{\mathsf{onechal}} < y$.

(Honest-Verifier) Zero-knowledge proof. Exactly as in [122], the final proof contains only one Σ-protocol transcript, and thus the HVZK is maintained (see the full version for details) [ToDo: fix].

Remark 14 (Network of machines with single-challenge ASICs.)**.** A sophisticated strategy that the adversary might attempt in order to bypass our protocol and encumber the key in a TEE is to utilize an outsourced network of machines, each equipped with an ASIC, in such a way that each ASIC is given only one challenge to solve. This ensures that no machine gets access to two challenges that would enable extraction of the witness. This strategy is highly impractical, however, since it requires the adversary to not only use a large number of outsourced machines, but also make strong non-collusion assumptions among them. In particular, if any two machines belong to the same entity or collude, they can reconstruct the witness which the adversary needs to avoid.

Remark 15 (Usage of sequential functions)**.** Our usage of hash-based PoW mining comes with some unfortunate consequences: since evaluation is embarrassingly parallel, we need to rely on assumptions on the parallel processing capabilities (e.g., number of machines) available to the adversary.

A natural question towards removing this constraint is whether we can leverage sequential computation (e.g., through verifiable delay functions or VDFs [36]) instead of parallelizable hash computations. This turns out to be somewhat tricky however since intermediate values may first be computed in a TEE following which the rest of the computation can be done in faster untrusted hardware. We leave the exploration of this direction to future work.

9.6.4 Practical ASIC-ZKPoCK Parameterization

Achieving both completeness and forced revelation introduces a tension in the tuning of d and τ. For completeness, ASIC-ZKPoCK requires moderately large k_{asic} (corresponding to Q_{asic}). To ensure forced revelation (specifically, to rule out use of CPUs) requires small k_{adv}. To better understand this tension, it is helpful to consider the ratio $N = Q_{\mathsf{asic}}/Q_{\mathsf{cpu}}$, i.e., the speed advantage conferred by an ASIC over a CPU. Given an adversary with a network of m CPUs, the ratio $k_{\mathsf{asic}}/k_{\mathsf{adv}} = (\tau Q_{\mathsf{asic}}/d)/(\tau n Q_{\mathsf{cpu}}/d) = N/m$. A secure and efficient (in terms of n and y) parameterization requires $m \ll N$, i.e., that an adversary cannot feasibly come close to approaching ASIC speeds with a network of CPUs.

Example practical parameterizations. We derive two concrete parameterizations for ASIC-ZKPoCK; one for a recent top-of-the-line mining ASIC [33] with $Q_{\mathsf{asic}} \approx 2^{47}$ H/s and one for a cheap, outmoded mining ASIC with $Q_{\mathsf{asic}} \approx 2^{43}$ H/s. The hashrate for a state-of-the-art CPU [212] is taken as $Q_{\mathsf{cpu}} \approx 2^{26}$ H/s. We show that $n = 8$ and $n = 12$ rounds respectively for the the two settings yields negligible completeness and soundness error against an adversary with a network of 10,000 CPUs. The full version contains further details.

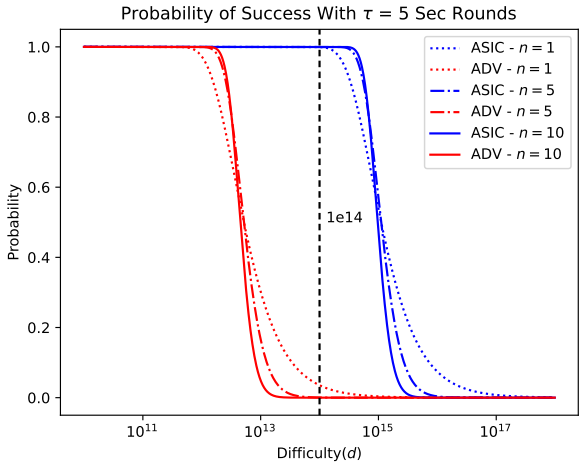

Figure 9.5: ASIC-ZKPoCK allows for practical parameters to achieve overwhelming probability of completeness by an ASIC (hashrate 154 TH/s) and negligible probability of success by an adversary with 10,000 state-of-the-art CPUs (hashrate 72 MH/s). n denotes number of rounds. Here, prover is required to compute more than $y = \lfloor n/2 \rfloor$ valid PoW puzzle solutions. The black line represents a parameterization of $d = 10^{14}$, for which adversarial success probability is negligible but that of ASIC is close to 1.

Smart Contract	LOC	Operation	Gas cost
ASIC-ZKPoCK Verification Contract	140	Register new job	366,485
		Initiate challenge	70,209
		Verify proof	7,620,401
CK Registry	130	Record a new proof	54,260

Figure 9.6: Lines of code (LOC) for SMACK and gas costs for various operations. As of Oct 2023, the cost for 100,000 gas is around $1.55.

Figure 9.5 compares honest and adversarial prover success rates for the top-of-the-line ASIC. We also implement an end-to-end system using the outmoded ASIC (see Section 9.7).

9.7 SMACK: End-to-End Implementation

To demonstrate CK proofs in a practical setting, we prototype SMACK (*SMArt-contract enabled CK*)—a *complete, end-to-end* CK system on Ethereum. SMACK offers a good proof of concept of CK practicality for two reasons. First, smart contracts are highly resource constrained, with limited, expensive computational power, coarse-grained, approximate measurement of time, and no ability to maintain secret state. Making CK proofs work in this austere environment strongly evidences their general practicality. Second, deployment of SMACK in Ethereum has the benefit of supporting a wide variety of blockchain-based services, e.g., voting, Atomic NFTs, enforced NFT royalties, etc., as described in Section 9.2.

SMACK allows for any desired CK method to be used. To demonstrate the practicality of our approaches, we implement two CK variants in SMACK: ASIC-ZKPoCK and a TEE-based CK proof system for Android devices that we call *lightweight CK* (details in the full version). Fig. 9.6 reports on the costs for SMACK's contracts.

CK Registry. SMACK supports CK for private keys associated with Ethereum addresses (public keys). CK proofs remain indefinitely valid (once an sk is exposed to $\mathcal{E}$, the fact of exposure remains true forever). Therefore we deploy a smart contract (at address 0x25B270...eE3966), called the CK *registry* that maintains a permanent record of addresses for which CK proofs have been provided (to the verification contract). This allows for applications to cheaply query whether a given address has an associated CK proof. As an example, we use this registry and verification contracts to mint Atomic NFTs (contract deployed

at 0x6986fc...FcAC38) for CK addresses. We provide more details on the registry in the full version.

9.7.1 ASIC-ZKPoCK implementation

We now detail our ASIC-ZKPoCK implementation, which is the more technically challenging and intricate of the two currently supported CK methods in SMACK. We make use of a Bitcoin miner that is several years old—the Bitmain Antminer S9 with a hashrate of 13TH/s. While this hardware cannot competitively mine Bitcoin at this point, it is sufficient to achieve low error bounds in our scheme[10] (details in the full version)[ToDo: fix]. It demonstrates that cheap, outmoded hardware (which cost us around $100) can be repurposed successfully for ASIC-ZKPoCK. We implement the verifier as an Ethereum smart contract, deployed at the address 0xAC86fD...B39f4b, and carry out an example verification in transaction 0xf67b1c...6f4106. All the source code and scripts can be found at `https://github.com/CK-anon/SMACK`.

System Architecture. Figure 9.7 shows our system architecture. The prover communicates with the ASIC miner using an open source pool mining software which implements the standard Stratum V2 protocol [112] for allocating work to the miner and fetching the PoW solution. The miner does some sanity checks on the input block data (e.g. for increasing block heights, correctly sized fields, etc.). Therefore, our prover makes use of a private Bitcoin network to generate valid PoW puzzles for the miner. This also allows us to configure the difficulty

[10] As $\tau = 12s$ for our implementation, any time advantage that an adversary can have in communication with the Ethereum blockchain does NOT meaningfully affect the prover's completeness probability.

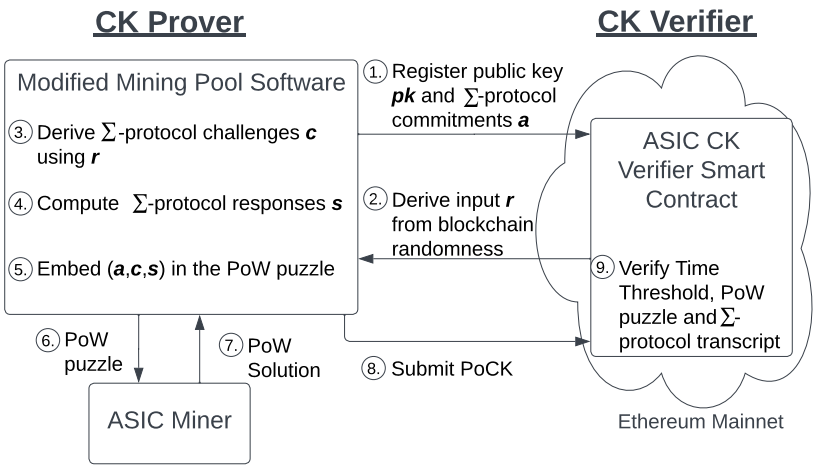

Figure 9.7: ASIC-ZKPoCK system architecture. The encircled numbers correspond to steps in a protocol execution. Our CK Prover consists of a (modified) pool mining software and an ASIC miner. Our CK Verifier is implemented as a Smart Contract on Ethereum mainnet.

of the PoW puzzles. We use Schnorr's protocol [226] for our underlying PoK Σ-protocol. While our verification smart contract is used for verifying complete knowledge of the private keys of Ethereum addresses, it can be used for any general PoK value x. To generate the verfier's random challenge r, our smart contract uses randomness from the Ethereum proof-of-stake network [113].[11] For pool mining, the Merkle root in the block header is expanded into a special coinbase transaction along with the adjacent Merkle branches. We place s (32 bytes in our case) inside this coinbase transaction, which can carry arbitrary data.

Setting the Nonce range β. For exploring the PoW puzzle solutions, the Bitcoin pool mining protocol allows miners to try different values for the *nonce* and *extranonce* fields. The nonce field in Bitcoin header is fixed to 4 bytes and has proven to be too small for the mining market. Therefore, extranonce was

[11] This randomness is biasable to small extent but is not material.

introduced whose size can be set by the pool software. Thus, for our case where the range is $\beta(> 2^{32})$, we set extranonce to $\lfloor log_2[\beta]/8 - 4 \rfloor$ bytes. Note that B in the proof transcript π denotes the portion of the Bitcoin block header that excludes the extranonce field, i.e. we treat that field as part of the space of possible nonces.

9.7.2 Lightweight CK

A PoCK system will be most useful if it is widely accessible. Any requirement for expensive specialized hardware—such as an SGX-enabled machine or, worse still, a Bitcoin-mining ASIC—could place CK beyond the reach of most users. While users could in principle outsource CK-proof execution, this would require them to entrust their private keys to third-party services, which could create new risks of key compromise.

Increasingly many users, however, *do* in fact own devices with trusted hardware: their mobile phones. Almost all newly manufactured mobile phones come with TEEs: recent Android devices often ship with Trusty TEE, while iOS devices have Apple's Secure Enclave.

To show how these devices can be used to implement CK proofs, we prototype a protocol design that we call *lightweight CK*. The term "lightweight" here reflects two features of our design: (1) It uses common consumer hardware, but (2) embodies a weaker security model than CK variants using SGX or ASIC. As such, lightweight CK is most suitable as a defense-in-depth layer or for applications where the impact of compromise is not high—e.g., Atomic NFTs (see Section 9.2.2).

Android implementation and workflow. We design a simple lightweight CK tool for Android devices that uses the hardware key attestation API [12] to produce lightweight CK proofs for Ethereum addresses. This API provides a hardware-backed assurance of boot integrity and, by extension, an application's integrity. (Apple's iOS analog is its App Attest Service [13], which can support a CK tool like the one we've implemented for Android.)

Our application itself is simple from a user's standpoint: the user enters a private key sk—exported from a crypto wallet—into a text field[12] and taps a button to generate a TEE key. The application copies the necessary CK proof, described below, to the Android clipboard for the user to paste into a dApp that creates a transaction to a CK verifier smart contract.

The app creates an attestation challenge for the freshly generated TEE key through the key attestation API, and the TEE signs a new certificate containing the challenge. The attestation challenge contains the signature of a static message ("Android CK Verification"), including an Ethereum message prefix [35] signed by the user's private key. While not strictly needed for a CK proof, the signature serves as a hedge against device compromise. Even if a compromised Android device could generate a seemingly valid integrity verdict, it cannot do so for an arbitrary pk—cooperation from the holder of the corresponding sk would be required.

The API then returns a certificate chain from the new TEE key containing the challenge to the TEE itself to the device manufacturer's certificate authority and, finally, to a root of trust—the Google Hardware Attestation Root Certifi-

[12]Great care will be required in guiding users, as malicious software could dupe unsophisticated users into revealing their private keys. This can be mitigated if common wallets natively implement lightweight CK.

cate [11]. Within the new TEE key's certificate is an attestation to the integrity of the operating system running on the device as well as a hash of the signing certificates of the application that made the request.

The certificate chain includes all the necessary information to create a CK proof, so submitting a lightweight CK proof to the Android CK Verifier smart contract involves creating a transaction that includes the intended prover address and the complete certificate chain to some root of trust. To ensure the authenticity of a lightweight proof, the contract first checks that the newly created certificate describes an adequately protected Android operating system: a verified boot from a trusted state and a key attestation for an app that matches the package name and signing key of our app. It also checks the attestation challenge embedded within the certificate to verify that sk signed the message. Next, it validates the certificate chain to a root certificate that the verifier trusts by verifying the signatures of each certificate, each one signed by the next in the chain. If everything passes validation, a record of the proof is created in the contract for use by the CK Registry. The cost of verification is approximately 1.5 million gas per certificate.

Signed messages cannot protect against a compromised device producing valid verdicts for others' keys, so we rely on per-device limits to mitigate the effect of a compromised device. Until mobile devices support on-device attestations, integrity measurements of the operating system are the closest way of verifying that the application ran as intended and a complete private key was entered into the device.

Limitations. Key attestation appears not to be foolproof. For example, there have been reports of a broken TEE keystore implementation in the ASUS ROG 3, compromising system integrity and still allowing a "strong" hardware-based integrity check to pass irrespective of bootloader status [252]. Google itself recommends a defense-in-depth approach, with attestation services as only one of several signals of abuse. To mitigate the problems caused by an entire class of devices containing faulty TEEs, the Android CK Verifier contract allows individual CA certificates to be revoked or trusted.

Privacy. Publishing a complete TEE certificate chain to a public blockchain comes with its own privacy issues. Each TEE certificate must be signed by a device manufacturer's public key for the certificate chain to be complete, which means that device manufacturers could easily associate the Ethereum address of a lightweight CK participant with the mobile device used to create the TEE certificate chain. This is because the Ethereum address being verified is contained within the TEE-signed certificate, and device manufacturers can associate a TEE's public key with its corresponding device both during manufacturing and whenever the device requires an updated certificate chain. As a point of reference, the first intermediate certificate of our sample device expires on a monthly basis, so at least once a month, the device must contact its manufacturer for a fresh intermediate certificate.

In order to prevent this type of privacy leak, rather than submitting the certificate chain directly to a smart contract, the certificate chain could instead be submitted to an application running within an off-device attesting TEE, such as Intel SGX. The attesting TEE application verifies the most sensitive part of the chain—including the mobile device TEE's public key—and the integrity state

of the mobile device embedded inside the leaf certificate. The smart contract would then only need to verify the application's attestation to establish whether the lightweight CK attempt was successful, thereby keeping the end of the certificate chain from being disclosed. Then, collusion between both the mobile device manufacturer and the organization hosting the off-device TEE, as well as a feasible attack on the off-device TEE itself, would be necessary to deanonymize accounts. An attack on the TEE could reveal the tail of the certificate chain which, when revealed to the device manufacturer, could be mapped to a device.

In an alternative to the TEE-based privacy approach, the prover's CK proof could be a redacted certificate chain along with a ZK proof (preferably a *succinct* such proof) of the correctness of the redacted part of the chain.

Target applications. Google's Play Integrity API is currently the more commonly used method for developing assurance of application integrity inside Android apps, and its "strong" category of integrity verdicts include similar hardware-backed key attestations from a TEE. Although its integrity verdicts can range in strength from basic compatibility tests to hardware-backed key attestation, it has seen wide use by consumer services (e.g., Netflix), mobile games (e.g., Pokémon Go), banking applications, etc. for application integrity [175].

Given the security limitations discussed above, lightweight CK is most suitable as a defense-in-depth layer or where the impact of compromise is limited. For example, if CK is compromised for an Atomic NFT, that NFT can be fractionalized: an undesirable but not catastrophic outcome. The same is true of key-coupling for royalty payments. In contrast, only strong CK would meet the levels of security envisioned for soulbound tokens [250], which are identity

documents.

Through the CK Registry, individual applications can support the specific CK proof types that match their security models.

9.8 Related Work

Concurrent work: Individual Cryptography [107]. In recent work concurrent to the first online version of this paper, Dziembowski et al. [107] propose the notion of *individual cryptography*, with a goal similar to ours—to prove that e.g., a key is held unencumbered by an individual party instead of within trusted hardware, or an MPC committee. This however is the extent of similarities. We highlight key differences below:

First, and most importantly our CK formalism and setting is more general—the main focus of their paper is on a simpler setting where the verifier also has access to the witness w—this does not model any of our usecases. Furthermore, [107] explicitly formalizes "slow" vs "fast" hash function computation as part of their primitive to model the computational difference between trusted (MPC or TEE) and untrusted (ASIC) computation; this reduces the scope of their primitive. In contrast, we provide a general resource-based formalism which captures other CK constructions—e.g., our SGX-based construction which *cannot* be modeled through their formalism. Our ASIC-based scheme also substantially differs from theirs. It further shows connections to Fischlin [122], allows the usage of generic Σ-protocols, and can be practically parameterized.

Second, while [107] positions their work as a theoretical exploration, we pro-

vide significant practical motivations: bribery attacks through key encumbrance on voting protocols, as well as new applications enabled by CK. Furthermore, we demonstrate practicality by not only implementing our constructions but also deploying them via Ethereum smart contracts to enable CK-based applications.

Preventing sharing of cryptogrpahic functionalities. Traitor-tracing [77] and leakage-deterring [165] schemes can both model the following abstraction: If an adversary with key sk attempts to share some cryptographic functionality (e.g., signing, encryption) that uses sk, then any entity that it shares this with could also recover sk. Note that this abstraction is orthogonal to CK; it could be feasibly used for voting but *does not apply to other CK applications*. If black-box recovery is possible, this can deter adversarial encumbrance (and bribery) since the user could always recover her own key.

Even then, these techniques do not capture all kinds of encumbrance we consider for several reasons: First, and most importantly, they are *purpose-built* for not only a specific functionality (e.g., signing) but also a specific scheme—given one scheme, a different scheme can be constructed that deters sharing as above. That is, they cannot *generically* prevent any type of encumbrance; in contrast, this is achieved by CK. Additionally, for signature schemes (required for voting) in particular, recovery is non-black-box. Second, they *do not prevent MPC-based encumbrance* where no party has the secret or a fully-operational functionality for even a single input. Finally, we note that requiring modification of the encryption/signature scheme would be a non-starter in the typical blockchain settings we consider where the voting key also controls, e.g., digital assets.

9.9 Conclusion

We have shown a fundamental limitation in traditional proofs of knowledge (PoKs)—the fact that they do not actually prove knowledge by a prover in light of encumbrance-based attacks using TEEs or MPC. This gap in the PoK model introduces a range of coercive attacks, many explored in earlier works. We therefore formalized complete knowledge (CK) as a stronger version of PoKs. CK can help in the design of practical protocols—using TEEs and ASICs—that are resistant to coercive attacks. We hope that our work will stimulate the development of new CK constructions and their use in e-voting, deniable authentication, lease-resistant credentials, and many other applications.

CHAPTER 10

SMART COLLUSION

10.1 Introduction

Cryptographic protocols often make *honesty* assumptions on how participants act. A common assumption is that fewer than t (out of a total of n) players are corrupt, while all other players execute the protocol honestly. This model is seen in many applications, including secret sharing [229], threshold cryptography [96], multi-party computation [136, 28], and private information retrieval [78].

But in practice, it is often more realistic to consider nodes as *rational* with potential *monetary incentives* to collude. As an example, in the context of secret sharing (where each player holds a share of a secret s), a coalition of players might profit by colluding to recover s earlier than expected. It is therefore important to design protocols to discourage rational players from colluding as they seek to maximize their utility.

In this work, we explore a broad framework for integrating techniques from both mechanism design and cryptography to establish feasibility and impossibility results for dealing with collusion.

Deterring collusion. Since we cannot make it impossible for players to collude, we will instead focus on *deterring* collusion by e.g., making it unstable or unprofitable for rational players. One promising approach is to provide a way for colluding players to *whistleblow* (or *snitch*) against their co-conspirators for a

reward. This was explored very recently for secret-sharing protocols in particular by Dziembowski et al. [108] and Gong et al. [138] (work concurrent to ours). In this context, [108] considers only the cryptographic angle for whistleblowing while [138] also formalizes how rewards should be set. Separately, other works build *cryptographic* tools for accountability (i.e., detecting malicious parties) for various primitives (see Section 10.1.2 for related works).

In this work, we design a broad framework for whistleblowing protocols that captures several applications. Abstractly, a whistleblowing protocol will consist of two components: (1) a *cryptographic* tool that enables players to verifiably prove the existence of collusion and/or its members; and (2) a *game-theoretic* mechanism to incentivize players to actually blow the whistle and impose penalties on colluders. The cryptographic proof fits within the broad umbrella of proving failures in the protocol's original security model. But without understanding the incentives, it *says nothing* about whether players will want to use it as opposed to staying silent.

We demonstrate how a strong yet natural form of collusion—which to the best of our knowledge has not been considered by prior work—can be used to *circumvent the positive results* from [108, 138]. Our work considers whistleblowing protocols in the context of this new form of collusion, which we will call *smart collusion*[1].

To avoid additional trust assumptions on who verifies the correctness of the collusion proof, we will require that our whistleblowing protocols not require trusted executors or private state. In other words, the verification can itself be instantiated through a blockchain smart contract.

[1]The term *smart* collusion comes from its similarity to the usage of blockchain smart contracts by colluding players to coordinate with each other in arbitrary ways.

A new notion — *smart* collusion. The basic idea behind smart collusion is that colluders have knowledge of the mechanism chosen by the protocol designer and can act accordingly in an attempt to circumvent the mechanism.

As an illustration, consider the following scenario. Suppose that we have a cryptographic protocol (e.g., secret sharing) that is susceptible to collusion. In an attempt to deter collusion, the protocol contains a cryptographic way for players to prove the existence of a collusion, and provides a reward R for submitting such a proof. But now, knowing the existence and details of this whistleblowing mechanism, colluders can adapt. In particular, all colluders can first agree on a contract C that holds large deposits from each player in the collusion and *retaliates* if anyone attempts to defect (from the collusion) by whistleblowing. As a simple example, retaliation might mean burning the deposit of the whistleblower if the mechanism is not anonymized or burning *all* deposits if it is. One potential way to instantiate C is as a blockchain smart contract.

In general, we will allow players to collude in arbitrary ways—both cryptographically to prevent the *ability* to whistleblow, and through designing threats of retaliation to deter players from whistleblowing. In essence, we note that for smart collusion, the same tools available to the protocol designer to protect against collusion will also be available to the colluders in their attempt to circumvent the protocol. Colluders in fact may have a *last-mover advantage* as the protocol designer must generally commit to their mechanism first. To the best of our knowledge, this perspective is missing from prior works on collusion in cryptographic protocols.

Smart collusion can be thought of analogously to real-world retaliation and intimidation tactics used to discourage defection from criminal enterprises. For

example, mafia bosses might threaten to harm or kill members (or their families) for violating the code of *omertà* (which mandates, among other principles, a refusal to cooperate with law enforcement). In game-theoretic language, smart collusion enables creating a "credible commitment" [198] to enforcing such penalties on defectors.

10.1.1 Our Contributions and Results

We make contributions on whistleblowing protocols from both the cryptographic and the mechanism design perspectives. We summarize our main results in this section.

Whistleblowing formalism (Section 10.3). We start by proposing a general design framework for whistleblowing protocols in Section 10.3. Abstractly, whistleblowing protocols comprise of a cryptographic tool to prove collusion, and a game-theoretic mechanism to incentivize players to submit such proofs. On the cryptographic side, we define security through typical notions of completeness (some player within the collusion can compute a collusion proof) and soundness (if there is no collusion, no player can compute a collusion proof). On the game-theoretic side, we define security in terms of incentivizing rational players to not collude. This requires ensuring that if players were to collude, they would be incentivized to defect by whistleblowing (i.e., negating the benefits of colluding). To capture this, we use the standard game-theoretic notion of a subgame-perfect pure Nash equilibrium (SPPNE).

Whistleblowing settings (Sections 10.4 and 10.5). We formalize two settings that capture several natural cryptographic applications and explore whistleblowing protocols for them.

1. *Blockchain-assisted transparent services (Section 10.4).* We model a generic distributed cryptographic service provided by a set $\mathcal{S}$ of servers. A key k is secret-shared among the servers. Any client can provide an input x and learn the result of some (potentially randomized) function $f(k, x)$ computed by the servers in a threshold manner. We illustrate how this models natural blockchain-based use cases such as threshold signing-as-a service, randomness beacons [104], and anonymous credentials [102] (see Section 10.4.4).

For many applications, it can be important to provide *transparency* (i.e., keep a public record of all answered queries) to prevent equivocation or enable auditability. As an example, for anonymous credentials that need to be rate limited, we want to ensure that *even a malicious server quorum* cannot provide extra credentials to a client. A natural first step is to use e.g., a blockchain, to log all queries (potentially in a privacy-preserving way). As an orthogonal point, using the blockchain for communication also provides natural *censorship-resistance* guarantees, similar to its use in [219, 75, 79].

However, even when queries are logged, notice that servers can still collude (individually or together) with a client to provide their service *out-of-band*, i.e., without communicating through the blockchain. A client may want to pay a premium for such out-of-band service that is not recorded (e.g., to receive extra credentials).

Our goal is to disincentivize this out-of-band collusion through whistleblowing. We accomplish this by ensuring that the client will be able to prove that it

has received service out-of-band—i.e., the client obtains a whistleblowing proof. We prove general feasibility results on when such whistleblowing proofs can be (cryptographically) constructed, and how they can be incentivized.

2. *General setting with collusion proofs (Section 10.5).* Second, we model a general setting that captures any protocol where proofs of collusion arise. That is, when players collude, some member necessarily obtains information that allows it to prove the existence of the collusion. Further, we even permit the collusion to decide the exact subset of colluding players who will receive such a proof.

While we don't have generic results for when cryptographic tools for proving collusion exist, we show how this fits several existing protocols including secret sharing with snitching [108, 138], threshold decryption with self-incriminating proofs [74], and BFT forensics [230].

The core reason for modeling this general setting is to show when such proofs of collusion are actually *effective*. We apply our game-theoretic framework and results to highlight the challenge of deterring collusion even given proofs of collusion (especially in light of our smart collusion notion).

Game-theoretic results (Sections 10.6 and 10.7). We formalize a general game-theoretic framework (capturing all our settings) that models the incentives of whistleblowing protocols (Section 10.6). More concretely, to deter collusion, we allow for designing a mechanism $\mathcal{M}^{\mathbb{W}}$ that takes deposit D from each player, and executes appropriate penalties / rewards when whistleblowing proofs are submitted. In turn, following our smart collusion notion, the colluders may themselves effectively write a "retaliation" mechanism $\mathcal{M}^{\mathbb{R}}$ that takes in deposit D' from each colluding player, and executes appropriate "counter-penalties"

based on whether whistleblowing proofs were submitted by defectors. For our game-theoretic analysis, we will abstract out the exact details of how whistleblowing proofs are constructed cryptographically. Our highlight results are as follows:

- *Impossibility of unconditional security*. We prove the impossibility of *unconditional* protection against smart collusion. In other words, for *any* whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$, there exists a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ that takes in sufficiently high deposit from colluders and succeeds in deterring them from blowing whistles. Perhaps surprisingly, we show this impossibility holds even when (1) the whistleblowing is *anonymous*, i.e., prevents identification of who is blowing the whistle; and (2) the whistleblowing is *fully accountable*, i.e., provably identifies *all* players in the collusion.

This can be seen in contrast with the positive results in e.g., [138] where standard collusion (i.e., without a retaliation contract) can be deterred when all players are rational. We show a similar feasibility for whistleblowing without retaliation in our broad framework.

Similarly, this impossibility highlights the challenge of *accountable* cryptography (e.g., [77, 39, 139, 38, 37, 41]) which often assumes trusted executors to identify malicious players. Our result may be interpreted as an impossibility for accountability in settings where this executor is rational and could collude with other players.

- *Small whistleblowing deposits can prevent collusion based on much larger retaliation deposits.* On the positive side, we show a concrete whistleblowing mechanism (with anonymous proofs) which requires a deposit D from each player and protects against any smart collusion with much larger deposits—up

to roughly $(t-1)D$ from each colluder who has the ability to whistleblow. This result has practical significance especially in settings where n (and therefore, t) are large.

- *Lower bound on the whistleblowing deposit.* Further, as a lower bound, we show that any whistleblowing mechanism that collects a whistleblowing deposit D can be rendered ineffective by some retaliation mechanism collecting a deposit $(t-1)\,D$. This demonstrates that our mechanism is near-optimal.

- *Positive result in the transparent service setting.* We find an interesting nuance that leads to a positive result for natural applications of the transparent service setting. Specifically, our result applies if the servers provide their service to clients on an ongoing basis.

Servers can be expected to let their whistleblowing deposit be held *indefinitely* (until a whistle is blown), effectively as a *cost of doing business*. But now, we show that the only way to break this whistleblowing mechanism is for the servers to force the client's deposit in the retaliation contract to also be held indefinitely. This would be a permanent cost to the client, which would intuitively negate any benefit from colluding to obtain off-band service. In turn, this disincentivizes collusion.

Chapter organization. The rest of the chapter is organized as follows: Section 10.1.2 presents related works. Section 10.2 gives relevant cryptography and mechanism design preliminaries. Section 10.3 introduces the broad framework for whistleblowing protocols; the cryptographic side is explored in Sections 10.4 and 10.5, and the mechanism design side is explored in Sections 10.6, 10.7, and 10.8.

10.1.2 Additional Related Works

Rational cryptography. A long line of work [126, 145, 15, 142] models the security of various cryptographic protocols under rational adversaries. Here, rationality is modeled in one of two ways: (i) either as a monolithic external attacker but with a budget (which is *weaker* than the typical arbitrary malicious model), or (ii) as all players being individually rational. Collusion amongst players is typically not considered. In our setting, we model individually rational players who can also *collude*, and moreover do so in a new powerful way with smart collusion.

Accountable protocols. Accountability—i.e., identification of malicious players—is a recurring theme in the cryptography literature. Accountable protocols have been put forth in a number of different contexts—traitor tracing [77, 257], threshold signatures (see e.g., [37, 40] and references therein), tracing for threshold decryption [38], detection of failures in trusted applications [9, 160], secret sharing [39, 139], threshold VRFs [41], and BFT forensics [230, 48].

But, typically, these protocols require that a trusted party executes some "accountability procedure" to identify the culprits (except for accountable threshold signatures). Even in cases where this executor does not require private state (such as a tracing key), if the executor is rational, it would still need to be *incentivized* to run this accountability procedure. In other cases (such as accountable signatures), similar to our whistleblowing design, individual parties or clients could still be required to submit their proofs to a public verifier. Consequently, smart collusion (with e.g., the executor) could prevent accountable protocols

from working as desired.

Collusion in the mechanism design literature. A sequence of works [1, 30, 25, 117, 194] study collusion-resistance in mechanism design. The literature broadly follows one of the following two approaches. In the first, colluding players merge into a single entity and maximize the coalition's joint utility [133, 72, 99, 141, 222]. In the other, collusion only modifies the payoffs in the game, but players continue to act individually to maximize their own utility [131, 211, 125, 120]. Our work aligns more closely with the latter.

Geffner and Tennenholtz [131] propose to let players place "bets" on information learned about others through collusion as a way to make collusion unstable. Schummer [227] considers bribes for taking specific actions within a collusion, but makes the trust assumption that the briber does not strategically misreport its preferences. In general, smart collusion is not captured since these works do not allow colluders to *extend their strategy space* and *respond* strategically to such anti-collusion mechanisms.

Another relevant thread is the literature on mechanisms with credible commitments [100, 149, 110, 121, 75]. These typically focus on commitments made by an auctioneer, but smart collusion instead uses them to strengthen the collusion.

A recent line of work [222, 125, 80] explores blockchain transaction fee mechanisms (TFMs) which aim to prevent collusion between the miner (auctioneer) and users. But results are largely negative even with standard collusion, and feasibility results using cryptography [231] still require honest majority. Credible commitments between colluders (similar to smart collusion) are not consid-

ered.

10.2 Preliminaries

10.2.1 Cryptographic Preliminaries

Definition 10.1 (NIZK-PoK)**.** A non-interactive proof system $(\mathcal{P}, \mathcal{V})$ for a relation R is said to be honest-verifier zero-knowledge proof-of-knowledge if it satisfies the following properties:

- **Completeness.** $\forall\ (X, w) \in R$**,**

$$\Pr[\mathcal{P}(X, w) \leftrightarrow \mathcal{V}(X) \implies 1] \geq 1 - \mathsf{negl}(\lambda)$$

- κ**-Knowledge soundness.** There exists an extractor $\mathcal{E}$ that runs in expected polynomial time, such that, for every PPT prover $\mathcal{P}^*$ and every statement X**,**

$$\Pr[(X, w) \in R : w \leftarrow\!\$\ \mathcal{E}^{\mathcal{P}^*}(X)] \geq \Pr[\mathcal{P}^*(X) \leftrightarrow \mathcal{V}(X) \implies 1] - \kappa$$

- **Honest-verifier zero-knowledge,** There exists an efficient simulator Sim such that

$$\forall\ (X, w) \in R, \{\mathsf{Sim}(X)\} \approx_c \{\mathsf{View}_{\mathcal{V}}(\mathcal{P}(X, w) \leftrightarrow \mathcal{V}(X))\}$$

  where $\mathsf{View}_{\mathcal{V}}$ is the verifier's view in the protocol execution.

10.2.2 Mechanism Design Preliminaries

We will be modeling the games induced by whistleblowing mechanisms as *extended form games*. Intuitively, an extended form game captures a multi-stage game, where each stage requires players to play an action from some action space. The game proceeds differently to the next stages based on different actions taken by the players in each stage of the game.

Definition 10.2 (Extended form game)**.** An extended form game consists of a directed tree (H, E), where H denotes the set of histories. For players $1, \ldots, n$, the game starts at the root with history $h = \emptyset$. For any given history h, the actions available to player i is given by the action space $A_i(h)$. For any actions $(a_1, \ldots a_n) \in \Pi_i A_i(h)$ taken by the n players at history h, the game transitions from history h to $h' = \mathcal{T}(h, a_1, \ldots, a_n)$ for some $(h, h') \in E$. The game terminates when the history h reaches a terminal vertex (i.e, there exists no $(h, h') \in E$). Player i then receives a utility $u_i(h)$ at termination.

Players are modeled as rational, i.e., each player selfishly tries to optimize its utility received at the end of the game. It is also *common knowledge* that all players are utility-maximizing.

Intuitively, in the simplest case where the game consists of only a single round (i.e, the root $h =$ is also a terminal node), an action profile $(a_1, \ldots, a_n) \in \Pi_i A_i(\emptyset)$ is a *Nash equilibrium* if no player i can increase their utility by deviating from playing a_i.

Definition 10.3 (Pure Nash equilibrium)**.** In a single-round game with action space $A_1, \ldots A_n$, an action profile $(a_1, \ldots, a_n) \in \Pi_i A_i$ is a pure Nash equilibrium if for all players i and actions $a_i' \neq a_i$ in A_i, i's utility $u_i(a_i, a_{-i})$ is at least

$u_i(a'_i, a_{-i})$, where a_{-i} is the action profile $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$ of the players other than i.

In a multi-stage extended form game, player i's strategy $s_i : H \to \bigcup_{h \in H} A_i(h)$ denotes the action it would take at any given point in history. A profile of strategies $(s_1, \ldots, s_n)$ is a subgame perfect Nash equilibrium if no player i can increase its utility by switching to a different strategy s'_i.

Definition 10.4 (Subgame perfect pure Nash equilibrium, SPPNE)**.** For a strategy profile $\vec{s} = (s_1, \ldots, s_n)$, let $\vec{s}|_h$ denote the strategy profile $\vec{s}$ restricted to the extended form game with h as the root. Then, $\vec{s}$ is a subgame perfect pure Nash equilibrium if, for all histories h reached by playing the strategy $\vec{s}$, $\vec{s}|_h$ is a pure Nash equilibrium.

We adopt SPPNE as the equilibrium concept for the games induced by the whistleblowing and retaliation mechanisms — we will analyze the induced games for the existence of SPPNEs.

10.3 Whistleblowing Framework

In this section, we introduce the basic design of whistleblowing protocols. We aim only to provide a general overview here; some details may be left underspecified and will be elaborated on in later sections for our specific settings.

Whistleblowing protocols. As discussed earlier, abstractly, a whistleblowing protocol will consist of two parts: (i) a cryptographic tool that allows a colluding party to prove existence of the collusion and/or its participants to a verifier

$\mathbb{V}$; and (ii) a game-theoretic mechanism that is allowed to take deposits from players, will essentially run the verifier $\mathbb{V}$ (to check valid proofs of collusion), and can execute rewards / penalties based on submitted whistleblowing proofs (or just *whistles*).

Definition 10.5 below provides a general, (informal) definition for whistleblowing protocols, which includes security goals for both the cryptographic and game-theoretic components.

We expand on the cryptographic part in Section 10.4 and the game-theoretic part in Section 10.6.

Definition 10.5 (Whistleblowing Protocol (Informal))**.** Consider a (ideal) functionality $\mathcal{F}$ among the set $\mathcal{P}$ of n players and consider $\mathbb{A} \subseteq 2^{\mathcal{P}}$ (intuitively denoting "allowed" adversarial corruption sets). An α-whistleblowing protocol for $(\mathcal{F}, \mathbb{A})$ is a tuple $(\Pi, \mathbb{P}, \mathbb{V}, \mathcal{M}^{\mathbb{W}})$ where Π is a protocol, $\mathbb{P} \leftrightarrow \mathbb{V}$ is a proof-system, and $\mathcal{M}^{\mathbb{W}}$ is a mechanism as follows.

(I. Cryptography) First, for the cryptographic properties, as is typical, we can model a *monolithic* adversary $\mathcal{A}$ to define security.

- (*Protocol validity or correctness*). Π emulates $\mathcal{F}$ in the presence of an adversary $\mathcal{A}$ that controls a set of players $\mathcal{P}_{\mathcal{A}} \in \mathbb{A}$.
- (*Whistleblowing completeness*) If $\mathcal{A}$ controls players $\mathcal{P}_{\mathcal{A}} \notin \mathbb{A}$ (e.g., $\mathcal{A}$ corrupted more players than expected), then there are at least α distinct players ($\in \mathcal{P}_{\mathcal{A}}$) that can convince $\mathbb{V}$.
- (*Whistleblowing soundness / unframability*) If $\mathcal{A}$ controls players $\mathcal{P}_{\mathcal{A}} \in \mathbb{A}$, then no player $P \in \mathcal{P}$ can convince $\mathbb{V}$.

Here, by "P convinces $\mathbb{V}$", we mean that the verification program $\mathbb{V}$ outputs 1 when interacting with P. Note that $\mathbb{V}$ will be provided with the public setup parameters from Π.

We say that the whistleblowing protocol is anonymous if $\mathbb{V}$ cannot distinguish which player submitted a valid proof. Further, it is β-*identifiable* if, as part of the proof to $\mathbb{V}$, it can correctly identify β players in $\mathcal{P}_{\mathcal{A}}$ (without framing honest ones in $\mathcal{P} \setminus \mathcal{P}_{\mathcal{A}}$).

Further details are given in Sections 10.4 and 10.5.

(II. Mechanism Design) Here, we will model each player as independent and rational. Each player can can choose one of two actions, "collude" and "honest" (intuitively, if P plays "collude", from the cryptographic perspective, P will be considered part of the adversary $\mathcal{A}$).

We allow the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ to take deposits from each player, view whistles that were submitted, and execute rewards / penalties. Following our smart-collusion notion, colluders themselves will be allowed to collude using a mechanism $\mathcal{M}^{\mathbb{R}}$ of their choice. Now, security is defined in the following sense:

- (*Not colluding is rational*) For all $\mathcal{M}^{\mathbb{R}}$ according to which collusion can take place, the subgame-perfect pure Nash equilibrium for the game induced by $(\mathcal{M}^{\mathbb{W}}, \mathcal{M}^{\mathbb{R}})$ is where everyone plays "honest."

Section 10.6 provides a more rigorous description.

Remark 16 (Processing rewards)**.** Rewarding whistleblowing proofs, even when they are anonymous, is straightforward with cryptocurrency. For exam-

ple, submission of a whistle can be accompanied with a fresh cryptocurrency wallet address (owned by the submitter) to which any reward from $\mathcal{M}^{\mathbb{W}}$ will be sent.

10.4 Transparent Service Model

We now describe a formal model for whistleblowing in our transparent service setting.

Basic model. We model three entities: a set of n service providers (or servers) $\mathcal{S}_1, \ldots, \mathcal{S}_n$, a user (or client) $\mathcal{U}$, and the logger $\mathcal{L}$ (e.g., a blockchain). The servers $\{\mathcal{S}_i\}$ collectively provide users with a computational functionality for a (potentially randomized) function f such that the user can learn $f(x)$ on input x by interacting with any t servers. For our formalism, since each user query is done separately, it is sufficient to only model one user. The goal for $\mathcal{L}$ is to (publicly) log *all* user queries (potentially in a privacy-preserving way)—intuitively, this will happen by parties communicating through $\mathcal{L}$. However, since the servers $\{\mathcal{S}_i\}$ can always communicate with users off-band (i.e., circumventing $\mathcal{L}$), the goal is to make it incentive-compatible to use $\mathcal{L}$. The cryptographic part of this is to ensure that the user can *whistleblow* by proving that it received off-band service.

We start by formally defining a generic computational service provided by the servers $\{\mathcal{S}_i\}$. This captures a broad range of applications, which we discuss in more detail in Section 10.4.4. In Section 10.4.2, we present a formal definition of a whistleblowing protocol for off-band collusion. Later, we characterize what

functionalities admit whistleblowing, and present concrete protocols.

10.4.1 A Generic Computation Functionality

Definition 10.6 (Computation Functionality Syntax)**.** Consider a (randomized) function $f : \mathcal{K} \times \mathcal{X} \times \mathcal{R}_S \to \mathcal{Y}$ over key space $\mathcal{K}$, input space $\mathcal{X}$, randomness space $\mathcal{R}_S$, and output space $\mathcal{Y}$. We write $(x, y) \in f_k$ if there exists r_s such that $f(k, x, r_s) = y$.

We focus on a threshold functionality, wherein the evaluation key is secret-shared among the n servers such that the client can communicate with any t servers to compute the function.

Formally, a computational functionality $\mathcal{F}$ between the servers $\{\mathcal{S}_i\}$ and a user $\mathcal{U}$ for computing f is a tuple $(\mathsf{Gen}, \mathsf{Query}, \mathsf{Respond}, \mathsf{Output}, \mathsf{PartialVer}, \mathsf{Ver})$ of algorithms as follows:

- $\mathcal{F}.\mathsf{Gen}(\lambda, t, n) \to (k_1, \ldots, k_n, \mathsf{vk})$ is a (randomized) key generation algorithm run either by a trusted party or by MPC between the servers $\{\mathcal{S}_i\}$. It generates a key pair $(k, \mathsf{vk}) \in \mathcal{K} \times \mathcal{K}_V$ along with the secret shares $\{k_i\}$ of the key k that will be stored by the servers $\{\mathcal{S}_i\}$.
- $\mathcal{F}.\mathsf{Query}(x; r^c) \to x^*$ is a (randomized) algorithm run by the client that takes as input $x \in \mathcal{X}$ and randomness r_c and outputs a leakage x^* of the input. We will formalize the notion of leakage later in this section.
- $\mathcal{F}.\mathsf{Respond}(k_i, x^*; r_i^s) \to (y_i^*, \pi_i)$ is a (randomized) algorithm run by the ith server $\mathcal{S}_i$ that takes as input the key share k_i, the leakage x^*, and randomness r_s and outputs an intermediate result y_i^* and a proof π_i.

- $\mathcal{F}.\mathsf{Output}(\{(y^*_{i_1}, \pi_{i_1}), \ldots, (y^*_{i_t}, \pi_{i_t})\}; r^c) \to (y, \pi)$ is a deterministic algorithm run by the client that takes as input the intermediate results $\{y^*_{i_1}, \ldots, y^*_{i_t}\}$ and randomness r_c, and computes the final output $y \in \mathcal{Y}$ along with a proof π. If the result is *self-verifiable*, the proof might be empty: $\pi = \emptyset$.
- $\mathcal{F}.\mathsf{PartialVer}(\mathsf{vk}, x^*, y^*_i, \pi_i) \to \{0, 1\}$ is a deterministic verification algorithm which takes as input vk, x^*, y^*_i, and a proof π_i and outputs a single bit denoting whether the response of the ith server was valid.
- $\mathcal{F}.\mathsf{Ver}(\mathsf{vk}, x, y, \pi) \to \{0, 1\}$ is a deterministic verification algorithm which takes as input vk, x, y, and a proof π and outputs a single bit denoting whether y is indeed the correct function evaluation at x, i.e. $(x, y) \in f_k$.

Leakage. Intuitively, x^*, y^*_i capture the "leakage" of the function – i.e., the values recorded by $\mathcal{L}$. For input x and randomness r^c, we will use $\mathsf{leak}_{\mathcal{F}}(x; r^c)$ to denote the output x^* of $\mathcal{F}.\mathsf{Query}(x; r^c)$. We further use $\mathsf{leak}_{\mathcal{F}}(x)$ to denote the distribution $\{\mathsf{leak}_{\mathcal{F}}(x; r_c)\}_{r_c \in \mathcal{R}_C}$ and $\mathsf{leak}_{\mathcal{F}}(X)$ to denote the joint distribution of $\mathsf{leak}_{\mathcal{F}}(x)$ for all $x \in X$. We often drop the subscript when $\mathcal{F}$ is clear from context.

Of particular interest to us will be the three classes of leakage functions, which we now describe.

- *Public.* A public leakage function reveals x to $\mathcal{S}$ and $\mathcal{L}$. Formally, $x^* = x$ and $\mathcal{R}_C = \perp$. This captures functionalities such as signatures, VRFs, decryption, and Identity-based encryption (IBE) enrollment (see Section 10.4.4 for details).
- *Oblivious Leakage.* Intuitively, oblivious means that the leakage function reveals nothing about x. This holds even if the adversary adv is allowed to see evaluations of the function at arbitrary inputs of its choice. We formalize this

as a standard indistinguishability game where adv chooses two inputs x_0, x_1, is given back $\mathsf{leak}(x_b)$, and wins if it correctly guesses b (see Definition 10.7).

Definition 10.7. A functionality $\mathcal{F}$ has $\epsilon(\lambda)$-bounded oblivious leakage if for all $\lambda \in \mathbb{N}$ and all adversaries $\mathcal{A}$ (making $\mathsf{poly}(\lambda)$ queries to $\mathcal{F}$), the following expression is bounded by $\epsilon(\lambda)$:

$$\mathbf{Adv}ob - leak\mathcal{F}[(\text{adv})] = \left| \Pr\left[(\mathcal{A}(l_b) \Rightarrow b') = b \;\middle|\; \begin{smallmatrix} b \leftarrow\$ \{0,1\} \\ x_0,x_1 \leftarrow \mathcal{A}(1^\lambda) \\ r \leftarrow\$ \mathcal{R}_C \\ l_b = \mathsf{leak}(x_b;r) \end{smallmatrix} \right] - \frac{1}{2} \right|.$$

Applications such as anonymous credentials and blind signatures have oblivious leakage. Later in this section, we will prove that unfortunately, it is impossible to whistleblow for such applications.

- *One-way function leakage.* This captures a weaker but still useful notion of privacy. Intuitively, the leakage function reveals some information about x, but not enough to recover x. Concretely, for a random input x, given $\mathsf{leak}(x)$, it is difficult for any PPT adversary to compute x. This is formalized in Definition 10.8.

Definition 10.8. A functionality $\mathcal{F}$ has one-way function leakage if, for all $\lambda \in \mathbb{N}$ and all PPT adversaries adv, the following advantage is negligible in λ:

$$\mathbf{Adv}ow - leak\mathcal{F}(\text{adv}) = \Pr\left[\mathcal{A}(l) \Rightarrow x \;\middle|\; \begin{smallmatrix} x,r \leftarrow\$ \mathcal{X}\times\mathcal{R}_C \\ l = \mathsf{leak}(x;r) \end{smallmatrix} \right].$$

While we do not find applications that inherently have such leakage, in Section 10.4.3.3, we show how we can support whistleblowing for applications (such as anonymous credentials) with oblivious leakage by weakening the privacy guarantee to one-way leakage.

Security properties. Security of the functionality $\mathcal{F}$ can be formulated as natural correctness and soundness properties.

Informally, correctness means that if all servers are honest, the client with input x will receive the correct output $f(x)$ and can verify this. Further, soundness requires that it should be hard for a client to compute the function at a new input $\hat{x}$, even if it can corrupt up to $t-1$ servers, and has oracle access to query other arbitrary inputs. This can be seen as a generalization of the typical *unforgeability* definition for e.g., signatures.

Since the ways to formalize these properties use relatively standard techniques, and our focus is instead on *whistleblowing* protocols for $\mathcal{F}$, we defer the full formal definitions to Appendix F.1.

10.4.2 Whistleblowing Formalism

Recall that our goal is to prevent off-band collusion between any t servers $\{\mathcal{S}_i\}$ and a client. To facilitate this, a first step is to allow the client to report off-band service provided by the servers $\{\mathcal{S}_i\}$. In this section, we detail formalism for how such *whistleblowing* can take place. Intuitively, a whistleblowing protocol will allow a user to convince the logger $\mathcal{L}$ exactly when it possesses a valid function evaluation (x, y) that was not recorded on $\mathcal{L}$. This can be used as an indication that at least t of the n servers are involved in providing off-band services. For simplicity, we focus on the functionalities where there is no way to detect which t out of n servers were responsible for providing off-band service. This fits several applications like (non-accountable) threshold signatures, VRFs, and IBE enrollment. Extending the formalism to identifiable whistleblowing (which applies to e.g., accountable threshold signatures) is straightforward (see Appendix F.1.)

Definition 10.9 (Whistleblowing for off-band service)**.** Functionality $\mathcal{F} = (\mathsf{Gen}, \mathsf{Query}, \mathsf{Respond}, \mathsf{Output}, \mathsf{Ver})$ cryptographically supports a whistleblowing protocol $\mathcal{W}_\mathcal{F}$ if its proof system $(\mathbb{P}, \mathbb{V})$ satisfies the following:

- *Completeness.* Intuitively, completeness means that a client with access to a valid function evaluation (x, y) such that a previous query for any x^* in $\mathsf{leak}_\mathcal{F}(x)$ was not logged by $\mathcal{L}$ (i.e., this was the result of off-band communication with $\mathcal{S}$), should be able to convince the verifier of this. More formally, for all $(k_1, \ldots, k_n, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda, t, n)$, $L \subseteq \mathsf{leak}_\mathcal{F}(\mathcal{X})$, $x \in \mathcal{X}$ such that $\forall r^c \in \mathcal{R}_C, \mathsf{leak}_\mathcal{F}(x; r^c) \notin L$ and y such that $(x, y) \in f_k$, the following holds:

$$\Pr\left[\mathbb{P}((L, \mathsf{vk}), (x, y)) \leftrightarrow \mathbb{V}(L, \mathsf{vk}) \Rightarrow 1\right] = 1$$

where the probability is taken over the randomness of $\mathbb{P}$.

- *Unframability.* Intuitively, unframeability guarantees that a client cannot falsely frame the servers, even if it colludes with upto $t - 1$ servers. Note that this is reasonable since it is hard for the client to compute the function on any input if less than t servers offer off-band service.

Definition 10.10. A whistleblowing protocol $\mathcal{W}_\mathcal{F}$ for a functionality $\mathcal{F}$ is said to be unframeable if the following is negligible in λ for all PPT adversaries adv:

$$\mathbf{Adv}unframe\mathcal{W}_\mathcal{F}[(\text{adv})] = \Pr[\mathbf{G}^{\text{unfram}}_{\mathcal{W}_\mathcal{F}}(\text{adv}) = 1]$$

where $\mathbf{G}^{\text{unfram}}_{\mathcal{W}_\mathcal{F}}$ is as defined in Figure 10.1.

We further say that $\mathcal{W}_\mathcal{F}$ is *anonymous* if the following holds:

- (Anonymity) For all $(k, \mathsf{vk}) \leftarrow \mathsf{Gen}(\lambda)$, $L \subset \mathsf{leak}_\mathcal{F}(\mathcal{X})$, $x_1 \neq x_2$ such that $\forall r^c \in \mathcal{R}_C, \mathsf{leak}(x_1; r^c) \notin L \wedge \mathsf{leak}(x_2; r^c) \notin L$, and y_1, y_2 such that $(x_1, y_1) \in f_k$ and

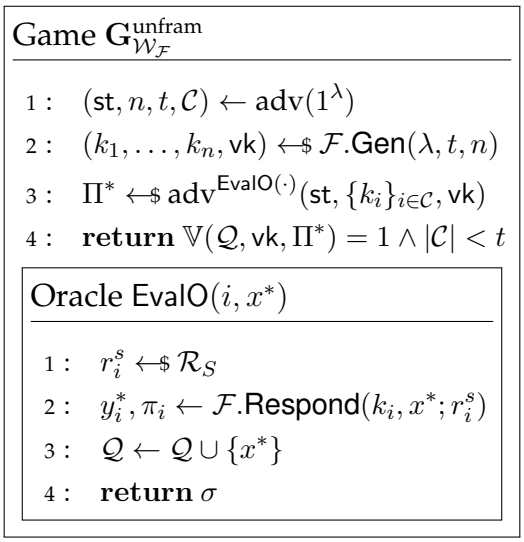

Figure 10.1: The unframability game for a whistleblowing protocol $\mathcal{W}_\mathcal{F}$ and a functionality $\mathcal{F}$.

$(x_2, y_2) \in f_k$, the two ensembles are computationally indistinguishable:

$$\{\text{View}_\mathbb{V}(\mathbb{P}((L, \mathsf{vk}), (x_1, y_1)) \leftrightarrow \mathbb{V}(L, \mathsf{vk}))\}$$

$$\approx_c \{\text{View}_\mathbb{V}(\mathbb{P}((L, \mathsf{vk}), (x_2, y_2)) \leftrightarrow \mathbb{V}(L, \mathsf{vk}))\}$$

where the distributions are over the randomness of the prover. Intuitively, anonymity implies that servers $\{\mathcal{S}_i\}$ learn nothing about which input was used for the whistleblowing.

10.4.3 Results on Whistleblowing Feasibility

We first look at whether whistleblowing is possible based on what is leaked (about the client query) to the servers $\{\mathcal{S}_i\}$ and the logger $\mathcal{L}$ by the functionality $\mathcal{F}$. Recall that $\mathsf{leak}_\mathcal{F}(x)$ denotes the distribution of the output $\mathcal{F}.\mathsf{Query}(x; r^c)$ over client randomness r^c.

10.4.3.1 Oblivious Leakage does not permit Whistleblowing

We show a simple result that whistleblowing (whether anonymous or not) is not possible for functionalities where the leakage is oblivious. Intuitively, this holds because oblivious leakage implies that no adversary adv can distinguish between $\mathsf{leak}_{\mathcal{F}}(x_0)$ and $\mathsf{leak}_{\mathcal{F}}(x_1)$. This implies that a user should be able to whistleblow with both x_0 and x_1 regardless of whether $\mathsf{leak}(x_0)$ or $\mathsf{leak}(x_1)$ was recorded on $\mathcal{L}$. But this contradicts the soundness property.

Lemma 10.11. *Consider any computational functionality $\mathcal{F}$ with leakage $\mathsf{leak}(\cdot)$ that is ϵ-oblivious (where $\epsilon = \epsilon(\lambda)$ is a negligible function). Then, no whistleblowing protocol with $(\mathbb{P}, \mathbb{V})$ can be ϵ-sound.*

The above lemma implies that it is impossible to support whistleblowing for functionalities like blind signatures and anonymous credentials (and their applications), which have oblivious leakage. A full proof is given in Appendix F.2. Nevertheless, in Section 10.4.3.3 we show how whistleblowing can be supported for some applications by weakening their privacy guarantee to one-way function leakage.

10.4.3.2 Whistleblowing with other leakages

We now present a protocol which works for *self-verifiable* functionalities with public leakage, as well as those with one-way leakage and small randomness space, i.e., $|\mathcal{R}_C|$ is polynomial in λ.

At a high level, if the client has a valid evaluation (x, y) for an input x not logged on $\mathcal{L}$, then it can use a NIZK-PoK to prove this to the logger. More

formally, let $0 < t < n \in \mathbb{N}$ and $(k_1, \ldots, k_n, \mathsf{vk}) \leftarrow\!\$\ \mathsf{Gen}(\lambda, t, n)$. Let L be the set of input leakages that have been recorded on the logger $\mathcal{L}$ so far. Suppose that a party is in possession of (x, y) such that $\mathsf{Ver}(\mathsf{vk}, x, y) = 1$ and $\forall r^c \in \mathcal{R}_C, \mathsf{leak}(x; r^c) \notin L$. Note that the Ver algorithm does not take a proof π as input since we are only considering self-verifiable functionalities wherein $\pi = \perp$ (we explain why we need this restriction in Remark 17). Now, to perform whistleblowing, the party computes a NIZK-PoK proof for the following relation and sends it to the verifier:

$$\mathcal{R} = \{(\mathsf{vk}, (x, y)) : \mathsf{Ver}(x, y, \mathsf{vk}) = 1 \wedge \forall r \in \mathcal{R}_C, \mathsf{leak}(x; r) \notin L\}$$

For the above relation to be polynomial-sized, we require the size of the randomness space $\mathcal{R}_C$ to be polynomial in the security parameter λ. This is trivially true for any functionality with public leakage, but this does restrict the functionalities with one-way leakage that we can support with the above protocol. We defer the full proof that this protocol is secure and anonymous to Appendix F.2.

Remark 17 (Generalizing to functionalities with verification proofs)**.** The above whistleblowing protocol works only for *self-verifiable* functionalities, i.e., where the proof π is empty, and it is possible to directly verify the evaluation pair (x, y) by just running the Ver algorithm. The reason is that there is a simple impossibility result against whistleblowing for functionalities where a special non-empty π is required. Specifically, the servers can choose to only provide the intermediate results y_i^* off-band, with either no proof, or just a designated verifier proof. This allows the client to compute the evaluation y, but not the evaluation proof. As a consequence, there is no efficient way for the client to prove to the logger $\mathcal{L}$ that y is indeed the correct evaluation at x.

10.4.3.3 Supporting whistleblowing for anonymous credentials by weakening privacy

In Section 10.4.3.1, we showed that it is impossible for functionalities with oblivious leakage to support whistleblowing. Nevertheless, we observe that for certain functionalities, it is possible to weaken the privacy guarantee to one-way leakage, without breaking the application. Specifically, our technique works for self-verifiable functionalities like blind signatures, if the input comes from a high-entropy space. This weakening, though not ideal, allows us to circumvent the impossibility result for blind signatures, which are used in anonymous credentials as well as privacy pass.

Specifically, we modify the functionality as follows. If the input has high entropy, we require the client to send a one-way function $z = \mathsf{owf}(x)$ of its input x in addition to the query $x^* \in \mathsf{leak}(x)$, along with a zero-knowledge proof of knowledge that it knows x, r^c used to generate x^* and z. More formally, it sends a NIZK-PoK of the following relation:

$$R_q = \left\{ ((z, x^*), (x, r^c)) : \begin{array}{c} x \in \mathcal{X}, r^c \in \mathcal{R}^C \wedge \\ x^* = \mathsf{leak}(x; r^c) \wedge z = \mathsf{owf}(x) \end{array} \right\}$$

.

Note that the servers only learn a one-way function of the client's input (and a zero-knowledge proof), and since we are considering an input space with high entropy, this functionality now has one-way leakage. Next, whenever the client wants to use the functionality output (x, y) (which is a blind signature in our example), it cannot simply reveal the tuple (x, y), because the servers can then trivially link this x back to one of the past queries by comparing $\mathsf{owf}(x)$ with the z values of past queries. We get around this by modifying how the clients use

their functionality output (x, y). Specifically, instead of directly revealing (x, y), the client can provide a non-interactive zero-knowledge proof of knowledge (NIZK-PoK) of such a tuple. This can also be extended to scenarios where the tuple (x, y) must be used only once (e.g. single-use tokens in privacy pass) by requiring the client to provide a different one-way function $\mathsf{owf}_2(x)$ of x when using the functionality output (x, y).

Lastly, the logger $\mathcal{L}$ will store the list L_z of $\{z = \mathsf{owf}(x)\}$ for all the inputs queried by the client, and the whistleblowing protocol can be run with respect to L_z, as described in Section 10.4.3.2.

If the input comes from a lower entropy space, then we can replace $\mathsf{owf}(x)$ (and $\mathsf{owf}_2(x)$) in the above technique to $\mathsf{owf}(x, r)$ (and $\mathsf{owf}_2(x, r)$), where r comes from a poly-sized space $\mathcal{R}$ (this is important for the efficiency of the whistleblowing protocol given in Section 10.4.3.2). This can be used to provide *concrete* security guarantees, in cases where it is possible to brute force the space of inputs $\mathcal{X}$ and randomness $\mathcal{R}$ separately, *but not together* (i.e., $\mathcal{X} \times \mathcal{R}$ is infeasible to brute force).

Note that this would not work for some applications of anonymous credentials such as PAKE and PSI, because leaking any one-way function of the inputs would trivially break the applications. We expand more on this in Appendix F.1.

10.4.4 Applications

We will now outline a number of natural applications that can benefit from whistleblowing to detect off-band collusion. While we describe them as dis-

tributed applications, note that clients will obtain similar collusion-proofs even with a single service provider.

Threshold signatures and variants. A threshold signature scheme allows n servers to sign a message only if at least t of them participate in the signing process. Signature validity can be directly verified by the client, with efficient schemes featuring constant signature size even as n increases.

Our framework can model a use-case where servers are supposed to provide a public signing service (e.g., a notary) via a blockchain. Here, off-band communication could allow the notary to equivocate on e.g., conflicting documents. Through whistleblowing, a client can easily submit an anonymous whistleblowing proof by showing that it knows a signature on a message that was not logged.

This setting can also be applied to other varieties such as private threshold signatures [37], accountable threshold signatures [139, 40] and weighted threshold signatures [130].

For the above, our whistleblowing protocol from Section 10.4.3.2 (with public leakage) will directly apply.

Verifiable random functions (VRFs) and distributed randomness beacons (DRBs). A VRF [190] is a pseudorandom function[2] whose output can be publicly verified. In a threshold VRF, the function evaluation key is secret shared among n parties, and a quorum of t parties is needed to evaluate the VRF. VRFs

[2]There is also the closely related notion of a verifiable unpredictable function (VUF) whose output is not necessarily pseudorandom but cannot be predicted completely.

can also be viewed as private threshold signatures with a unique signature for every message. VRFs can be used in transparency systems to support privacy by obscuring user names [189] or domain names [134] while ensuring they are uniquely mapped to a pseudorandom identifier. Threshold VRFs can also be used to construct distributed randomness beacons [76], as deployed in practice by Drand [104]. For applications to function correctly, it is crucial that this randomness is not leaked through e.g., off-band collusion. Our whistleblowing protocol (for public leakage) can be directly used here. Note however, that some VRFs are not self-verifiable (for example [41, 94]). Hence, our techniques will only apply to certain VRFs [104, 146].

Threshold Blind signatures and anonymous credentials. A threshold blind signature scheme is an interactive protocol involving a client and n servers with shares of a signing key, wherein the client can get a message signed by any t servers, but without revealing the contents of the message or signature to the servers. Blind signatures were originally proposed for anonymous digital cash [69] and have since been proposed for many other applications (see [124]), but we focus on the use case of servers issuing *anonymous credentials* [102]. Such systems allow users to obtain credentials privately, and later prove possession of credentials without revealing any other information about themselves.

Often, applications can necessitate that this issuance is somehow rate-limited (e.g., for Privacy Pass [95]). For this, it is important to ensure that servers do not collude with the client off-band to issue it extra credentials. Our whistleblowing framework would apply here. However, as the name might suggest, blind signatures (and in turn, anonymous credentials) have oblivious leakage—i.e., leak no information to the servers about the message being signed. Fortu-

nately, we can use the technique described in Section 10.4.3.3 to support whistleblowing by weakening the privacy guarantee to one-way leakage instead of obliviousness.

Oblivious PRFs. Oblivious PRFs (OPRFs) [63] are similar to blind signatures, except the output is the PRF evaluation of the input instead of a signature. Unlike signatures, PRFs do not necessarily support efficient proofs of correct evaluation. OPRFs are useful anywhere a hash function is used that should be rate-limited, for example in password-based encryption or for contact discovery in messaging applications via private set intersection (PSI). While OPRFs have oblivious leakage, we can support whistleblowing for self-verifiable OPRF as a service by relaxing the privacy to one-way leakage as before. However, for many applications that use OPRFs—such as private set intersection or password authenticated key exchange—this weakening of privacy could result in trivial breaks. We highlight an example in Appendix F.1.

10.5 General n-party Model

We now describe our second model, which applies to general n-party protocols with whistleblowing support. Specifically, we model a set of n parties $\mathcal{T}_1, \ldots, \mathcal{T}_n$ running a protocol Π (e.g., secret sharing or threshold decryption), wherein (i) security of the protocol holds as long as less than t parties collude; and (ii) if $\geq t$ parties collude, some of them get access to a publicly verifiable proof of the existence of collusion. Abstractly, we can divide Π into three sub-protocols:

- $\Pi.\mathsf{Gen}(1^\lambda, t, n) \to (k_1, \ldots, k_n, \mathsf{vk}, \mathbf{F})$ is a randomized protocol run between

a trusted dealer and all the parties. A distributed procedure could also be used. The output of the protocol includes secret shares $\{k_i\}$ of a secret value k to be stored by the parties $\{\mathcal{T}_i\}$, along with a public verification key vk which will be used to verify collusion proofs and $\mathbf{F}$, which characterizes the set of functions with the key k as input, that the parties are allowed to compute.

- $\Pi.\mathsf{Output}(f, k_{i_1}, \ldots, k_{i_t}) \rightarrow y$ is a (potentially randomized) protocol executed between any t parties, that takes as input a function description $f \in \mathbf{F}$, and outputs $f(k)$ where k is the value secret-shared among the parties.
- $\Pi.\mathsf{Ver}(\mathsf{vk}, \pi) \rightarrow \{0, 1\}$ is a deterministic algorithm that takes as input a proof of collusion π, and outputs a bit denoting its validity. Note that this collusion proof is publicly verifiable, and hence can be checked by a public smart contract.

Security properties (informal). In this model, a protocol with whistleblowing support, analogous to our description in Definition 10.5, will satisfy three properties: correctness, whistleblowing completeness, and unframability. We provide only informal descriptions below. These are relatively straightforward generalizations of properties from [108] to arbitrary protocols. Our focus in this paper is instead to highlight how other applications also fit this whistleblowing framework.

Correctness states that if all the parties are honest, then the Output protocol outputs the correct value, i.e. $f(k)$.

Whistleblowing completeness guarantees that if a set of atleast t parties collude

to learn some information about the secret k, then, at least one of the colluding parties can generate a publicly verifiable proof to either prove the existence of a collusion, or implicate another colluding party. This generalizes the punishability notion in [108].

Unframability can be formalized in two ways. The first applies to protocols wherein there is no way to detect which parties took part in the collusion. For such protocols, unframability requires that even if upto $t-1$ parties collude, they should not be able to produce a valid proof of collusion. The second notion applies to protocols where the collusion proof identifies atleast one colluding party. In this scenario, unframability states that even if upto $n-1$ parties collude with the trusted dealer (if any), they cannot blame any honest party for collusion. The formal definition for both notions are direct extensions of the unframability notion defined in Section 10.4.

10.5.1 Applications

We highlight several protocols that adhere to the above model. Our goal is to unify these different applications under our umbrella of whistleblowing, enabling the application of our game-theoretic insights to all of them.

Secret sharing with snitching. A secret sharing scheme allows storing sensitive information among n parties such that any t of them can reconstruct the secret. Secret sharing is widely used for storage, MPC, as well as threshold cryptography. In all of these applications, any t parties can collude to learn something about the secret. Recently, there has been work on developing secret

sharing schemes with whistleblowing capabilities [108, 138], such that at least one of the colluders learns a publicly verifiable proof implicating a colluding party. This is directly captured by our model, with the function space $\mathbf{F}$ containing just the identity function $f(x) = x$.

Threshold decryption. A threshold decryption scheme shares a decryption key among n parties such that any t of them can decrypt a ciphertext. It has applications in voting [90], auctions [225] and more recently, encrypted mempools in several blockchain projects [27, 201, 97] to protect against frontrunning MEV attacks [92]. But these protections are lost if t or more decrypting parties collude. While standard threshold decryption has no way to identify or prove that a collusion occurred, there has been recent work [74] on building threshold decryption schemes wherein if any t parties collude to illegally decrypt a ciphertext, at least one of them learns a proof of collusion. Collusion here is different from secret sharing, since the colluders only learn the decryption of a ciphertext, and not the full secret key. Our framework can be used to model this, with the function space $\mathbf{F}$ being the decryption for any ciphertext.

BFT forensics. Byzantine fault tolerant (BFT) consensus is a core primitive in the distributed systems literature. Intuitively, consensus protocols ensure that a distributed set of n servers agree on the same execution state. Typically, in asynchronous networks, these protocols require that the adversary corrupts upto $t' < n/3$ servers. Sheng et al. [230] show how some consensus protocols [256, 4] can support strong *forensics*—even if the adversary corrupts more than t' (but upto $2t'$) servers, the transcript of a single server can be used to detect adversarial servers. A similar model is also explored in [48] for permissionless protocols.

In our context, such a transcript can function as a whistleblowing proof of collusion. Note that the adversarial coalition may employ smart collusion to prevent servers from revealing their transcript to the forensics procedure.

10.6 Game-Theoretic Model

We formally model the incentives associated with collusion and whistleblowing in this section. We consider a set of n players $1, \ldots, n$ participating in a generic functionality where the players can receive an additional utility of V per player through collusion. Our core model in this section formalizes (n, n)-collusion — all n out of the n players need to collude in order to earn the reward nV. Section 10.8 extends the model to (t, n)-collusion (where it suffices that t players collude) and the transparent service setting.

Our goal is to design whistleblowing mechanisms that deter players from colluding and forming coalitions. Formally, we want to ensure that *all* subgame perfect pure Nash equilibria in the whistleblowing model involves players not colluding.

Model and game progression. We describe three (increasingly layered) models to arrive at the final model induced by the whistleblowing mechanism and smartly-colluding players.

1. The *base game* $\mathbb{B}$ considers only the incentives from colluding in the absence of whistleblowing — either all players collude and receive a utility V per player or some player remains honest, in which case, each player receives a zero

utility.

2. The *whistleblowing game* $\mathbb{W}$ models collusion in two stages when the protocol designer implements a whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ to counteract collusion — players decide whether to collude or not in the first stage, and whether to snitch on each other and submit whistleblowing proofs (aka blow whistles) to $\mathcal{M}^{\mathbb{W}}$ in the second stage. This would result in slashing the players for colluding and reward the whistleblowers.

3. The *retaliation game* $\mathbb{R}$ extends $\mathbb{W}$ by letting the players agree upon a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ that could penalize players for ratting each other out to $\mathcal{M}^{\mathbb{W}}$ and could also redistribute the whistleblowing rewards amongst them in case some player submits a whistleblowing proof. This is our final model for smart collusion.

Results. Before delving into the details of the three layers, we provide a brief overview of our results. We argue that *no* whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ can provide unconditional economic security — there always exists some retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ such that players can reach a subgame perfect pure Nash equilibrium by colluding in the first stage. Intuitively, $\mathcal{M}^{\mathbb{R}}$ could collect a deposit $D^{\mathbb{R}}$ orders of magnitude larger than the whistleblowing rewards and the gains V from colluding and threaten to burn all of them even if a single whistle is blown in $\mathcal{M}^{\mathbb{W}}$. This would nudge players to collude, but not blow any whistles since they would end up with an extremely negative utility otherwise. However, such a retaliation mechanism would require an extremely large deposit to be collected from the players upfront. We investigate the natural goal of designing whistleblowing mechanisms that are resilient to retaliation mechanisms that collect a deposit of up to $D^{\mathbb{R}}$ per player.

We formally describe the three layers next.

10.6.1 Base game $\mathbb{B}$

First, we consider the simple single-shot game $\mathbb{B}$ with no whistleblowing protocol in place, where players choose either to collude or to remain honest. If all players choose to collude, then each player receives a utility V. Otherwise, everybody receives zero utility.

Formally, each player i plays an action $b_i \in \{\text{collude}, \text{honest}\}$ for $1 \leq i \leq n$. Player i's utility function is then given by

$$U_i^{\mathbb{B}}(b_1, \ldots, b_n) = \begin{cases} V & b_1 = \cdots = b_n = \text{collude}, \\ 0 & \text{otherwise.} \end{cases}$$

Note that all players colluding is a pure Nash equilibrium in the base game $\mathbb{B}$ — changing their action from collude to honest will only decrease their utility from V to 0. Further, it is also the utility-maximizing outcome for the n players.

10.6.2 Whistleblowing game $\mathbb{W}$

Next, we consider the game $\mathbb{W}$ induced by the protocol designer implementing a whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ to deter collusion.

When the n players collude in the base game, suppose that some k of them learn a whistleblowing proof π. These k players can submit the whistleblowing proofs to $\mathcal{M}^{\mathbb{W}}$ (i.e., blow whistles) for a reward. To disincentivize players from colluding, the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ should slash the n players upon

receiving whistles. However, the rewards and slashes should be designed so that players are not discouraged to submit whistles in the first place, which could happen if the disutility from the slash far outweighs the utility from the reward. Note that smart collusion based retaliation is not considered at this stage.

The whistleblowing mechanism. We begin by discussing the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ before describing the game it induces. Formally, $\mathcal{M}^{\mathbb{W}} = (\mathbf{P}^{\mathbb{W}}, \mathbf{S}^{\mathbb{W}}) : \mathbb{N} \cup \{0\} \to \mathbb{R}^2_{\geq 0}$ is a map from the number w of whistles blown to the reward (or the payout) $\mathbf{P}^{\mathbb{W}}(w)$ disbursed per whistle and the slash $\mathbf{S}^{\mathbb{W}}(w)$ per player.

We model $\mathcal{M}^{\mathbb{W}}$ to have anonymous and simultaneous whistleblowing. Anonymity implies that the whistleblowing rewards and slashes cannot depend on the identities of the whistleblowers.[3] Simultaneity ensures that players decide on their actions at the same time—e.g., they decide the number of whistles to submit without knowing the decisions of the rest[4]. Note that anonymous whistleblowing implies that multiple whistles must be supported since the mechanism cannot distinguish between e.g., a single player submitting two whistles, and two players each submitting one whistle.

The reward $\mathbf{P}^{\mathbb{W}}(w)$ and slash $\mathbf{S}^{\mathbb{W}}(w)$ must satisfy the following natural con-

[3] Anonymity does not rule out the rewards and slashes depending on the *order* in which the whistleblowing mechanism receives the whistles. However, for simplicity, we assume that all n players have similar computational power and network connectivity, and thus, all possible orders of the whistles reaching $\mathcal{M}^{\mathbb{W}}$ are equally likely. Then, $\mathbf{P}^{\mathbb{W}}(w)$ can be thought of the expected reward per whistle (expectation taken over the randomness in network conditions) upon w whistles getting blown.

[4] If strategy leakage through e.g., the public blockchain mempool is of concern, timed cryptography primitives such as time-locked encryption or VDFs can be used to effectively make players act simultaneously. A similar modeling choice is made in [138].

ditions. To begin, whistleblowers cannot be penalized for submitting whistles — since whistles can be submitted from pseudonymous identities (e.g., fresh blockchain addresses), whistleblowers cannot be forced to pay any penalties enforced by $\mathcal{M}^{\mathbb{W}}$.

We also require $\mathcal{M}^{\mathbb{W}}$ to satisfy *budget balance* — the total slash collected should cover the rewards disbursed to whistleblowers. In other words, $w\mathbf{P}^{\mathbb{W}}(w) \leq n\mathbf{S}^{\mathbb{W}}(w)$ for all $w \in \mathbb{N} \cup \{0\}$. Note that this implies $\mathbf{S}^{\mathbb{W}}(w) \geq 0$. Indeed, $\mathcal{M}^{\mathbb{W}}$ should neither reward nor slash anyone when no whistles are blown, and thus $\mathbf{P}^{\mathbb{W}}(0) = \mathbf{S}^{\mathbb{W}}(0) = 0$.

To ensure that the players pay the slash charged to them, the whistleblowing mechanism collects a deposit $D^{\mathbb{W}}$ upfront from all the players. The deposit must be large enough to cover the slash irrespective of the number of whistles blown, and thus, $D^{\mathbb{W}} = \sup_w \mathbf{S}^{\mathbb{W}}(w)$. Our goal is to design "effective" whistleblowing mechanisms that collect as little deposit as possible.

Definition 10.12 (Whistleblowing Mechanism $\mathcal{M}^{\mathbb{W}}$)**.** A whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ is given by the pair $(\mathbf{P}^{\mathbb{W}}, \mathbf{S}^{\mathbb{W}}) : \mathbb{N} \cup \{0\} \to \mathbb{R}^2_{\geq 0}$ such that upon receiving w whistles, the perpetuator of each whistle is awarded $\mathbf{P}^{\mathbb{W}}(w)$ and all players are slashed $\mathbf{S}^{\mathbb{W}}(w)$, satisfying $w\mathbf{P}^{\mathbb{W}}(w) \leq n\mathbf{S}^{\mathbb{W}}(w)$ and $\mathbf{P}^{\mathbb{W}}(0) = \mathbf{S}^{\mathbb{W}}(0) = 0$. $D^{\mathbb{W}} = \sup_w \mathbf{S}^{\mathbb{W}}(w)$ is said to be the deposit of the mechanism $\mathcal{M}^{\mathbb{W}}$.

Remark 18 ($\mathcal{M}^{\mathbb{W}}$ is public)**.** We model $\mathcal{M}^{\mathbb{W}}$ to be *public*, i.e., it reveals the *total number* of whistles w that were blown (player identities can still be hidden). This is reflective of $\mathcal{M}^{\mathbb{W}}$ being instantiated as a (public) smart contract, and will only make it easier for colluders, later on, in the retaliation context.

For a given number k of players that learn the whistleblowing proof, there

is some natural upper bound on the number of whistles that they can blow within the time frame of the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$.[5] Reflecting this, we consider r-whistle mechanisms, that rewards nothing and slashes the entire deposit $D^{\mathbb{W}}$ upon receiving $r+1$ whistles or more. Essentially, the players would receive a net utility $-D^{\mathbb{W}}$ from blowing $r+1$ or more whistles, which is at most the utility that they would receive from blowing r whistles or less. Thus, the players would choose to blow at most r whistles even if they could submit more.

Definition 10.13 (r-whistle mechanism)**.** An r-whistle mechanism $\mathcal{M}^{\mathbb{W}} = (\mathbf{P}^{\mathbb{W}}, \mathbf{S}^{\mathbb{W}})$ is a whistleblowing mechanism where $\mathbf{P}^{\mathbb{W}}(w) = 0$ and $\mathbf{S}^{\mathbb{W}}(w) = D^{\mathbb{W}}$ for all $w > r$.

The extended form game. We now describe the extended-form two-round whistleblowing game $\mathbb{W}$ that models a whistleblowing protocol as seen by the n players. Prior to the start of the two rounds, players learn the utility V from colluding. The protocol designer then announces the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ and collects the whistleblowing deposit $D^{\mathbb{W}}$ from all players.

1. The first round of $\mathbb{W}$ is identical to the base game $\mathbb{B}$. Player i plays an action $b_i \in \{\mathsf{collude}, \mathsf{honest}\}$, to receive a utility

$$U_i^{(\mathbb{W},1)}(b_1, \ldots, b_n) = U_i^{\mathbb{B}}(b_1, \ldots, b_n).$$

The game proceeds to the second round if all n players collude and terminates immediately after the first round otherwise.

[5] Even if the players create many pseudonymous identities to blow whistles, this is not truly costless and would consume non-zero time (and in practice, non-zero transaction fee). This upper bounds the number of whistles that each player can submit within the time frame in which $\mathcal{M}^{\mathbb{W}}$ is eliciting whistles.

2. In the second stage of $\mathbb{W}$, the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ elicits whistles from the k players that know the whistleblowing proof (for convenience, reindex players so that these k players are labelled $1, \ldots, k$). For $1 \leq i \leq k$, player i's action space in the second round consists of blowing $w_i \in \mathbb{N} \cup \{0\}$ whistles. Players $k+1, \ldots, n$ each blow $w_i = 0$ whistles. Then, when $w = \sum_{i=1}^{n} w_i$ whistles are blown, player i receives a whistleblowing reward equal to $w_i \mathbf{P}^{\mathbb{W}}(w)$ and is slashed $\mathbf{S}^{\mathbb{W}}(w)$ for a total second stage utility

$$U_i^{(\mathbb{W},2)}(w_1, \ldots, w_n) = w_i \mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w).$$

Across the two stages, player i earns a net utility $U^{\mathbb{W}}(b_1, \ldots, b_n, w_1, \ldots, w_n)$ given by the sum of the utilities in the two stages.

For notational convenience, we denote the SPPNE $(\mathsf{collude}, \ldots, \mathsf{collude}), \vec{w}$ by $\vec{w}^{\mathbb{2}}$.

Subgame perfect pure Nash equilibrium in $\mathbb{W}$. The protocol designer would want to ensure that all SPPNEs of the whistleblowing game $\mathbb{W}$ consist of some player i playing $b_i = \mathsf{honest}$ in the first stage, while the coalition would want to play an equilibrium where all players $\mathsf{collude}$ (if one exists). We will begin our analysis of the players' incentives in $\mathbb{W}$ by characterizing all SPPNEs of $\mathbb{W}$.

A typical approach to computing all SPPNEs is through a bottom-up dynamic program. The players can be expected to play a pure Nash equilibrium $\vec{w} = (w_1, \ldots, w_n)$ in the second round of $\mathbb{W}$, since $\mathbb{W}$ terminates at the end of the round irrespective of their actions. Then, for a given second-stage equilibrium $\vec{w}$, the utilities from playing $\vec{w}^{\mathbb{2}} = (\mathsf{collude}, \ldots, \mathsf{collude}), \vec{w}$ and from playing honest can be compared. $\vec{w}^{\mathbb{2}}$ is a SPPNE if and only if all players i receive a

utility $U_i^{\mathbb{W}}(\vec{w}^2)$ at least zero, the utility from deviating and playing honest in the first stage (causing the game to terminate).

For $\vec{w}$ to be a second-stage equilibrium, no player i should be able to achieve a larger utility $U_i^{(\mathbb{W},2)}$ from deviating and blowing w_i' whistles instead of w_i for all $1 \leq i \leq k$. Formally, $\vec{w}$ is a pure Nash equilibrium in stage two if and only if

$$\begin{aligned} & w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) \\ & \geq w_i' \mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) \end{aligned} \tag{10.1}$$

for all players $1 \leq i \leq k$ and $w_i' \in \mathbb{N} \cup \{0\}$.

For some second-round equilibrium $\vec{w}$, to verify whether (collude, ..., collude), $\vec{w}$ is a SPPNE, we will first calculate the cumulative utility $U_i^{\mathbb{W}}$(collude, ..., collude, $\vec{w}$) received by each player i.

$$\begin{aligned} U_i^{\mathbb{W}}(\vec{w}^2) &= U_i^{(\mathbb{W},1)}(\text{collude}, \ldots, \text{collude}) + U_i^{(\mathbb{W},2)}(\vec{w}) \\ &= V + w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j). \end{aligned}$$

For $\vec{w}^2$ to be a SPPNE, each player i should receive a non-negative utility. Otherwise, player i will guarantee itself a zero utility by being honest in the first stage. In other words,

$$V + w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) \geq 0, \tag{10.2}$$

for all $1 \leq i \leq n$.

In summary, to ensure that all SPPNE of $\mathbb{W}$ involve some player being honest, the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ must ensure that either Equation (10.1) is violated for some $1 \leq i \leq k$ or Equation (10.2) is violated for some $1 \leq i \leq n$ for all possible strategy profiles $\vec{w}$ of the players in the second round.

Example. We will construct the "1-whistle skeleton" mechanism $\mathcal{M}^{\mathbb{W}}$ and argue that all SPPNE involve some player playing honest.

At a high level, for a collusion reward of V per player, the 1-whistle skeleton collects a deposit $V + \frac{\varepsilon}{n-1}$ from each player and slashes all of it even if a single whistle is blown.[6]

The whistleblower is rewarded $V + \varepsilon > V + \frac{\varepsilon}{n-1}$ if exactly a single whistle is blown, but disburses no reward if two or more whistles are submitted to $\mathcal{M}^{\mathbb{W}}$.

- $\mathbf{P}^{\mathbb{W}}(w) = \begin{cases} V + \varepsilon & \text{if } w = 1, \\ 0 & \text{otherwise.} \end{cases}$
- $\mathbf{S}^{\mathbb{W}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ V + \frac{\varepsilon}{n-1} & \text{otherwise.} \end{cases}$

Theorem 10.14. *For any $1 \leq k \leq n$, the 1-whistle skeleton mechanism $\mathcal{M}^{\mathbb{W}}$ described above deters collusion, i.e, all subgame perfect pure Nash equilibria in the whistleblowing game $\mathbb{W}$ involves some player playing honest in round one.*

Proof. We will calculate the bottom-up dynamic program by first spotting all second-stage pure Nash equilibria $\vec{w}$ (so that Equation (10.1) holds for all $1 \leq i \leq k$) and arguing that some player ends up with a negative utility for all such $\vec{w}$ (i.e, Equation (10.2) is violated for some player i).

We will argue that $\vec{w} = (0, \ldots, 0)$ does not satisfy Equation (10.1) for any of the k players that can blow a whistle. Indeed, by blowing no whistles,

[6] Think of ε as any small positive constant—orders of magnitude smaller than other discussed quantities. We only use it to clean up our analysis by breaking ties between the utilities of different actions taken by the players.

$U_i^{(\mathbb{W},2)}(0,\ldots,0) = 0$, but by deviating and blowing a whistle, player i increases its utility to

$$1 \times \mathbf{P}^{\mathbb{W}}(1) - \mathbf{S}^{\mathbb{W}}(1) = (V + \varepsilon) - (V + \frac{\varepsilon}{n-1}) > 0.$$

Next, we will argue that for any $\vec{w} \neq (0,\ldots,0)$, Equation (10.2) cannot be simultaneously satisfied for all n players, and thus, someone would want to deviate and play honest in the first stage. Consider some player i that receives no whistleblowing reward in the second stage. Such a player always exists — if exactly one whistle is blown, consider some player other than the whistleblower and if more than one whistle is blown, no player receives a whistleblowing reward. $U_i^{(\mathbb{W},2)}(\vec{w}) = -(V + \frac{\varepsilon}{n-1})$ and thus, $V - (V + \frac{\varepsilon}{n-1}) < 0$, violating Equation (10.2).

Thus, all SPPNE in the whistleblowing game $\mathbb{W}$ induced by the 1-skeleton mechanism involve some player playing honest in the first round. □

Note that if the deposit $D^{\mathbb{W}}$ collected by $\mathcal{M}^{\mathbb{W}}$ is less than V, the n players may choose to collude without concern of the second-stage outcome of $\mathbb{W}$, since they are slashed at most V in $(\mathbb{W},2)$. Hence, collecting a whistleblowing deposit V from each player can be a natural baseline to have in mind for all the discussions to follow.[7]

[7] Our lower bounds do not rule out a whistleblowing mechanism with a deposit smaller than V that deters collusion. Even though all players might be better off colluding in the first stage irrespective of the outcome in the second, there might not exist a pure Nash equilibrium in $(\mathbb{W},2)$. We defer a more detailed discussion to Appendix F.3.

10.6.3 Retaliation Game $\mathbb{R}$

In the two-round whistleblowing game $\mathbb{W}$, the protocol designer had to announce the whistleblowing mechanism before the start of the first round. This gives the players an opportunity to smartly collude by launching their own *retaliation mechanism* $\mathcal{M}^{\mathbb{R}}$ to *break* the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ — the retaliation mechanism looks to modify the payoffs of the second stage so that blowing $(w_1, \ldots, w_n)$ whistles is both, a second stage equilibrium and leaves every player with a positive net utility across the two rounds.

In particular, note that the second-stage equilibrium need not prevent every playing from blowing any whistles. The retaliation mechanism can choose to induce a different equilibrium as long as each player profits from colluding. For example, it could actually ask colluders to whistleblow and split their rewards as long as the utility for this state is larger than not-colluding.

The retaliation mechanism. We describe the retaliation mechanism before formally setting up the retaliation game. The retaliation mechanism $\mathcal{M}^{\mathbb{R}} = (\mathcal{M}_1^{\mathbb{R}}, \ldots, \mathcal{M}_n^{\mathbb{R}})$, where $\mathcal{M}_i^{\mathbb{R}} : \mathbb{N} \cup \{0\} \to \mathbb{R}$ maps the number w of whistles submitted to the whistleblowing mechanism to the penalty charged to player i by the retaliation mechanism. We use the term "penalty" quite loosely here — apart from penalizing players for not playing according to the coalition's strategy, the retaliation mechanism also serves the dual purpose of redistributing the whistleblowing rewards received by players $1, \ldots, k$ to $k+1, \ldots, n$, who cannot blow a whistle. Thus, without loss of generality, we can assume that the players $k+1, \ldots, n$ are charged a negative penalty $\mathcal{M}_i^{\mathbb{R}}(w) \leq 0$ (i.e, rewarded by $\mathcal{M}^{\mathbb{R}}$), irrespective of the number w of whistles blown. Otherwise, such a player is

strictly better off by being honest than colluding.

Similar to the whistleblowing mechanism, we require $\mathcal{M}^{\mathbb{R}}$ to satisfy budget balance. The retaliation mechanism is only a means to penalize players and to redistribute rewards amongst themselves. There cannot be new capital injected from an external source. Thus, we assert $\sum_i \mathcal{M}^{\mathbb{R}}(w) \geq 0$ for all $w \in \mathbb{N} \cup \{0\}$. Additionally, to ensure player i pays the penalty charged to it, it will have to submit a *retaliation deposit* equal to the largest penalty charged to i by $\mathcal{M}^{\mathbb{R}}$. Indeed, if a player is never penalized but is only rewarded by the retaliation mechanism, they do not have to submit a deposit. Formally, the retaliation deposit $D_i^{\mathbb{R}} = \max\{\sup_w \mathcal{M}_i^{\mathbb{R}}(w), 0\}$.

Definition 10.15 (Retaliation mechanism)**.** The retaliation mechanism $\mathcal{M}^{\mathbb{R}} : \mathbb{N} \cup \{0\} \to \mathbb{R}^n$ takes as input the total number of whistles w submitted by the players $1, \ldots, k$ to the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ and outputs the penalty imposed on each of the n players satisfying $\sum_{i=1}^{n} \mathcal{M}_i^{\mathbb{R}}(w) \geq 0$ and $\mathcal{M}_i^{\mathbb{R}}(w) \leq 0$ for all $i > k$. The deposit of the retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ is given by $D_i^{\mathbb{R}} = \max\{\sup_w \mathcal{M}_i^{\mathbb{R}}(w), 0\}$.

Unlike the whistleblowing mechanism, the retaliation mechanism need not be anonymous. $\mathcal{M}^{\mathbb{R}}$ can suggest different players to submit a different number of whistles to $\mathcal{M}^{\mathbb{W}}$, and can penalize them differently if a different number of whistles is observed.

Remark 19 (Executing $\mathcal{M}^{\mathbb{R}}$)**.** We model the retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ to be "self-executing" in the sense that it can observe the public output of $\mathcal{M}^{\mathbb{W}}$ and execute appropriate penalties without requiring external intervention. The reader might observe here that a smart-contract instantiation of $\mathcal{M}^{\mathbb{R}}$ would require a transaction to trigger it—this "relays" the public information about $\mathcal{M}^{\mathbb{W}}$ to $\mathcal{M}^{\mathbb{R}}$.

Even in this case, there is a natural, folklore technique to make it effectively self-executing. In particular, by offering a small reward to anyone in the world (not restricted to the n players) who relays this public information from $\mathcal{M}^{\mathbb{W}}$ to $\mathcal{M}^{\mathbb{R}}$, it can be guaranteed that the penalties in $\mathcal{M}^{\mathbb{R}}$ will execute.

The extended-form game. The retaliation game $\mathbb{R}$ proceeds similar to $\mathbb{W}$, except that players receive an additional penalty $\mathcal{M}_i^{\mathbb{R}}(w)$ in the second stage from blowing w whistles. We formally describe how the retaliation mechanism modifies the whistleblowing game $\mathbb{W}$. Similar to $\mathbb{W}$, the retaliation game $\mathbb{R}$ also occurs over two stages.

Apart from learning the collusion V, hearing about the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ and locking in the whistleblowing deposit $D^{\mathbb{W}}$, the players also agree upon a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ and put down the necessary retaliation deposits $D_1^{\mathbb{R}}, \ldots, D_n^{\mathbb{R}}$ before the first round commences.

The progression between the two rounds of $\mathbb{R}$ is identical to $\mathbb{W}$.

1. The first round consists of the base game $\mathbb{B}$, where players can play actions $b_1, \ldots, b_n \in \{\mathsf{collude}, \mathsf{honest}\}$ to receive a utility

$$U_i^{(\mathbb{R},1)}(b_1, \ldots, b_n) = U_i^{\mathbb{B}}(b_1, \ldots, b_n).$$

The retaliation game $\mathbb{R}$ proceeds to the second stage if and only if all n players $\mathsf{collude}$ in the first stage.

2. As with the whistleblowing game, the second-round action space for players $1 \leq i \leq k$ consists of submitting w_i whistles to $\mathcal{M}^{\mathbb{W}}$, while the remaining players blow $w_i = 0$ whistles. The whistleblowing mechanism then awards each player i with a utility $w_i \mathbf{P}^{\mathbb{W}}(\sum_i w_i) - \mathbf{S}^{\mathbb{W}}(\sum_i w_i)$. Additionally, they are also

charged a penalty $-\mathcal{M}_i^{\mathbb{R}}(\sum_i w_i)$ by the retaliation mechanism, to receive a total utility equal to

$$U_i^{(\mathbb{R},2)}(w_1,\ldots,w_n) = w_i\mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) - \mathcal{M}^{\mathbb{R}}(w)$$

in the second stage, where $w = \sum_{i=1}^n w_i$.

The total utility $U_i^{\mathbb{R}}(b_1,\ldots,b_n,w_1,\ldots,w_n)$ received by player i equals the sum of utilities from the two stages.

Similar to the whistleblowing game $\mathbb{W}$, we will abuse notation to denote the SPPNE $(\mathsf{collude},\ldots,\mathsf{collude}),(w_1,\ldots,w_n)$ by $(w_1,\ldots,w_n)$ by $(w_1,\ldots,w_n)^{\mathbb{2}}$.

Subgame perfect pure Nash equilibria in $\mathbb{R}$. As with the whistleblowing game $\mathbb{W}$, we locate all SPPNEs of $\mathbb{R}$ by a bottom-up dynamic program. $\vec{w}^{\mathbb{2}}$ is a SPPNE, $\vec{w}$ played by $1,\ldots,k$ must be a second-stage equilibrium, which implies

$$\begin{aligned} &w_i\mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) - \mathcal{M}^{\mathbb{R}}(\sum_j w_j) \\ &\quad\geq w_i'\mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j\neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j\neq i} w_j) - \mathcal{M}^{\mathbb{R}}(w_i' + \sum_{j\neq i} w_j) \end{aligned} \tag{10.3}$$

for all $w_i' \in \mathbb{N}\cup\{0\}$ and $1\leq i\leq k$. Further, $\vec{w}^{\mathbb{2}}$ is a subgame perfect pure Nash equilibrium if and only if each player i is better off than deviating and playing honest in the first stage.

$$\begin{aligned} U_i^{\mathbb{R}}(\vec{w}^{\mathbb{2}}) &= U_i^{(\mathbb{R},1)}(\mathsf{collude},\ldots,\mathsf{collude}) + U_i^{(\mathbb{R},2)}(\vec{w}) \\ &= V + w_i\mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) - \mathcal{M}_i^{\mathbb{R}}(\sum_j w_j) \\ &\geq 0 \end{aligned} \tag{10.4}$$

for all $1\leq i\leq n$.

The coalition aims to induce some SPPNE $\vec{w}^2$ in $\mathbb{R}$ where all players collude in the first stage. In such a case, we say that the retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ and the SPPNE $\vec{w}^2$ *breaks* the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$.

Breaking *any* whistleblowing mechanism through retaliation. For a given whistleblowing deposit $D^{\mathbb{W}}$, we will argue that any whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ can be broken by a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with a sufficiently large deposit $D^{\mathbb{R}}$. Intuitively, if the retaliation mechanism collects a deposit much larger than the utility from collusion and the whistleblowing rewards, $\mathcal{M}^{\mathbb{R}}$ can threaten to burn the entire $D_i^{\mathbb{R}}$ for every player i even if a single whistle is blown. Players are better off receiving zero utility in the second stage from blowing no whistles than incurring the massive retaliation penalty from blowing whistles. We expand on our intuition in Appendix F.5.1.

Theorem 10.16. *For any whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$, there exists a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with a sufficiently high deposit $D^{\mathbb{R}} \geq (n-1) \cdot D^{\mathbb{W}}$ such that $\mathcal{M}^{\mathbb{R}}$ and $(0, \ldots, 0)^2$ breaks $\mathcal{M}^{\mathbb{W}}$.*

This suggests that whistleblowing mechanisms cannot provide *unconditional economic security* against retaliation-enabled smart-collusion. For a fixed whistleblowing deposit $D^{\mathbb{W}}$, a natural objective for the protocol designer would then be to maximize the retaliation deposit required to break $\mathcal{M}^{\mathbb{W}}$.

We propose that the economic security of a whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ is characterized by the maximum retaliation deposit submitted by any player in the optimal retaliation mechanism that breaks $\mathcal{M}^{\mathbb{W}}$. This best captures the largest "financial stress" placed by $\mathcal{M}^{\mathbb{R}}$ on the n players in their attempt to break $\mathcal{M}^{\mathbb{W}}$. Accordingly, we define the economic security of $\mathcal{M}^{\mathbb{W}}$ as

$\min_{\mathcal{M}^{\mathbb{R}}:\mathcal{M}^{\mathbb{R}} \text{ breaks } \mathcal{M}^{\mathbb{W}}} \left(\max_i D_i^{\mathbb{R}}\right)$.

10.7 Effective Whistleblowing

10.7.1 The 1-Whistle Mechanism $\mathbb{M}_{1*}^{\mathbb{W}}$

We extend our 1-whistle skeleton to be resilient against retaliation mechanisms with deposit up to $D^{\mathbb{R}} = (n-1)\,\gamma$ for $\gamma \geq 0$.

We propose the mechanism $(n-1)\,\gamma$-secure $\mathbb{M}_{1*}^{\mathbb{W}}$, which collects a whistleblowing deposit $D^{\mathbb{W}} = V + \gamma + \frac{\varepsilon}{n-1}$ from each player and transfers the entire collected deposits to the whistleblower if a single whistle is blown. If two or more whistles are blown, then the entire deposit is slashed and none of the whistleblowers receive any reward. Formally, our mechanism is given by

1. $\mathbf{P}^{\mathbb{W}}(w) = \begin{cases} nV + n\,\gamma + \varepsilon & \text{if } w = 1, \\ 0 & \text{otherwise} \end{cases}$
   for some small $\varepsilon > 0$, and,
2. $\mathbf{S}^{\mathbb{W}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ V + \gamma + \frac{\varepsilon}{n-1} & \text{otherwise.} \end{cases}$

We argue that $\mathbb{M}_{1*}^{\mathbb{W}}$ cannot be broken by any retaliation mechanism that collects a retaliation deposit of up to $(n-1)\,\gamma$. Intuitively, when a single whistle is blown, $\mathbb{M}_{1*}^{\mathbb{W}}$ transfers greater than $V + \gamma$ units of money from all the players to the whistleblower. The $(n-1)$ players other than the whistleblower need to be paid strictly greater than γ per player for their net utility across the two

rounds to be non-negative (they receive V from collusion in the first round and are slashed greater than $V + \gamma$ in the second). However, the retaliation deposit is at most $(n-1)\gamma$, and thus, the whistleblower cannot be forced to redistribute more than $(n-1)\gamma$ units of cash.

As a consequence, there exists some player that receives a negative utility from colluding, and therefore, $\mathbb{M}^{\mathbb{W}}_{1*}$ cannot be broken with a retaliation deposit at most $(n-1)\gamma$.

Theorem 10.17. *The mechanism $\mathbb{M}^{\mathbb{W}}_{1*}$ cannot be broken by any retaliation mechanism that collects a deposit of at most $(n-1)\gamma$ from each of its players, irrespective of the number of players k that learn the whistleblowing proof.*

We defer the proof to Appendix F.5.2.

Observe that the retaliation deposit needed to break $\mathbb{M}^{\mathbb{W}}_{1*}$ scales linearly with the number of players. To paint a picture, for $n = 1000$, an additional $\$1000$ deposit from each player on top of the necessary deposit V will result in it being resilient to retaliation mechanisms with deposits of up to roughly a million dollars!

In Theorem 10.16, we argued that any whistleblowing mechanism that collects a deposit $V + \gamma$ can be broken by a retaliation mechanism with a deposit $(n-1)(V+\gamma)$. We explore tighter lower bounds in the next section.

10.7.2 Lower Bounds

We first argue that the $\mathbb{M}^{\mathbb{W}}_{1*}$ is optimal when $k = 1$, i.e, only a single player knows the whistleblowing secret (proof in Appendix F.5.3).

Theorem 10.18. *Any whistleblowing mechanism that collects a whistleblowing deposit of at most $V + \gamma$ can be broken by a retaliation mechanism with a retaliation deposit of at most $(n-1)\gamma$ and some strategy profile $\vec{w}^2$ when $k = 1$.*

Next, we argue that $\mathbb{M}_{1*}^{\mathbb{W}}$ is optimal for $k > 1$ amongst all mechanisms that (a) have a pure Nash equilibrium in the second stage and (b) the pure Nash equilibrium is such that the joint utility of the coalition across the two rounds is non-negative. A formal analysis is given in Appendix F.5.4.

We motivate assuming the existence of a second-stage pure Nash equilibrium satisfying the condition on net positive utility for the coalition in Appendix F.3.

10.7.3 Alternative Collusion Models

In Appendix F.4, we consider other interesting models of collusion. In particular, we consider models where players can set up *jointly controlled* accounts to further simplify collusion. We find that our results do not change even with this advantage given to colluders.

Further, we note that if desired, the whistleblowing protocol designer could prevent joint accounts through recent cryptographic techniques [156, 107] that allow proving *individual* ownership of keys.

We also show that non-anonymity of the whistleblowing mechanism does not change our result (except in the case that the whistleblowing deposit can be held indefinitely; see Section 10.8.2).

10.8 Beyond (n, n)-Collusion

In this section, we discuss designing whistleblowing mechanisms beyond (n, n)-collusion. Specifically, we discuss details for (t, n)-collusion and the transparent service setting.

10.8.1 (t, n)-Collusion

We find that from the game-theoretic perspective, (t, n)-collusion is surprisingly similar to (n, n)-collusion.

(t, n)-collusion generalizes the (n, n)-collusion environment as follows — it suffices for a coalition to contain any t out of the n players to successfully collude and derive a utility of V per player in the coalition. For a given coalition of players $1, \ldots, t$, we discuss designing whistleblowing mechanisms to deter smart-collusion.

Observe that the whistleblowing and retaliation games of (t, n)-collusion are identical to that of (t, t)-collusion, except for the budget balance constraint. Rather than only using the whistleblowing slashes from just the t colluders to disburse whistleblowing rewards, the whistleblowing mechanism can also rely on the slashes collected from the $n - t$ players that are not part of the coalition. Thus, budget balance in the (t, n)-collusion environment requires that $w\mathbf{P}^{\mathbb{W}}(w) \leq n\mathbf{S}^{\mathbb{W}}(w)$ for all $w \in \mathbb{N} \cup \{0\}$.

It is fairly straightforward to see that our $\mathbb{M}_{1*}^{\mathbb{W}}$ mechanism (from Section 10.7.1) continues to remain resilient against retaliation mechanisms that

collect a deposit of up to $(t-1)\,\gamma$.

Proposition 10.19. *The $\mathbb{M}^{\mathbb{W}}_{1*}$ mechanism cannot be broken by any retaliation mechanism with a retaliation deposit of at most $(t-1)\,\gamma$ in the (t,n)-environment, irrespective of the number of players $1 \leq k \leq t$ that learn the whistleblowing proof.*

Lower bounds in the (t,n)-collusion setting can also be derived analogously. We defer the details to Appendix F.5.5.

Identifying colluders. While some cryptographic whistleblowing proofs can identify a subset of the t players that choose to collude, we note that our impossibility result on unconditional economic security continues to hold. In fact, it holds even if the whistleblowing mechanism exactly knows which t out of the n players collude.

To see why, observe that even with (n,n)-collusion, the protocol designer knows that all n players have to collude in order to receive the collusion reward V (i.e., the whistleblowing proof "identifies" all colluders), yet no mechanism could be resilient against retaliation mechanisms with a sufficiently large retaliation deposit.

10.8.2 Transparent Service Setting

In this section, we design whistleblowing mechanisms for the transparent service provider model (with n servers and a client) from Section 10.4. The induced whistleblowing and retaliation games will mirror our analysis from earlier. We discuss some differences below.

We focus on collusion forming between t servers and the client (for off-band service). The client will learn the whistleblowing proof in such a case. While one or more server could also learn this, note that this will not yield additional utility to the coalition. One differentiating point is that whistleblowing mechanism can only collect deposits from servers and not from clients.

A very similar proof to Theorem 10.17 will show that an appropriate adaptation of our $\mathbb{M}_{1*}^{\mathbb{W}}$ mechanism continues to remain resilient against any retaliation mechanism that collects a retaliation deposit of up to $n\gamma$ from the client and the servers (for $n = 1000$ and $\gamma = 1000$, imagine a commercial service asking its client to lock in roughly a million dollars to use the service). Similarly, it is also not hard to see that our result on the impossibility of unconditional economic security carries over.

Nevertheless, we find an interesting positive result for unconditional security by leveraging a characteristic specific to this setting.

Positive result with *transient* clients. A natural application of the transparent service setting is one where servers provide their functionality as a *long term* service, while clients are transient and only appear when they wish to submit a query. This provides an interesting tool to the whistleblowing protocol designer—deposits from servers can be held *indefinitely* (essentially as a "cost of doing business"), or until there is a whistleblowing proof submitted. In turn, to break this mechanism and successfully provide off-band service to a client, the servers would also have to hold a retaliation deposit from the client indefinitely. Otherwise, the client could simply wait until its deposit is returned before whistleblowing.

However, it is likely that no client will want to have its retaliation deposit be held indefinitely in order to collude off-band. As a consequence, in our formalism, this translates precisely to the setting with $D^{\mathbb{R}} = 0$ (i.e., whistleblowing without a retaliation contract). This allows us to provide unconditional security.

Proposition 10.20. *There exist whistleblowing mechanisms that collect a deposit* $D^{\mathbb{R}} = V + \frac{\varepsilon}{n-1}$ *that cannot be broken by any retaliation mechanism in the transparent service model with long term service and transient clients.*

Conclusion

We have introduced a broad framework for *whistleblowing* with the goal of deterring collusion in cryptographic protocols. Whistleblowing protocols combine cryptographic tools for colluders to defect by ratting on each other, with mechanism design techniques to incentivize such defection.

We presented the new notion of smart collusion (i.e., collusion using blockchain smart contracts) and showed how it significantly boosts the colluders power. We showed generic cryptographic and game-theoretic results on the feasibility of whistleblowing.

Finally, we showed several existing and new cryptographic applications that can be captured in our whistleblowing framework, and would benefit from our analysis.

CHAPTER 11

CLOSING REMARKS

Broadly, this dissertation explored the challenges of designing security protocols in the blockchain setting in light of novel applications and powerful new attacker capabilities.

The first part provided a new perspective on transaction ordering within consensus protocols with important applications to decentralized financial systems. It defined the new order-fair consensus primitive, and built protocols with the goal of restricting adversarial manipulation of transaction ordering.

The second part showed how attackers can themselves employ trusted hardware, multi-party computation, and smart-contracts to break security. Chapter 9 showed powerful new *encumbrance* attacks, including on coercion-resistant voting and deniability in end-to-end messaging. It provided new insights on how knowledge is defined in cryptography and formulated the new *proofs of complete knowledge* primitive. Chapter 10 showed how blockchain smart-contracts can be used by malicious players to form stronger collusions and circumvent results in accountable cryptography.

11.1 How Blockchains Can Inform Modern Systems Security

The highly adversarial nature of blockchains can teach us a lot about designing secure systems more broadly. Modern systems are becoming increasingly complex with many moving parts. Our security models formulated many decades ago may no longer apply in the same way. Therefore, when designing secure protocols, it is important to first think carefully about *what* security means, and

how we define it. I envision that a multi-pronged, intersectional approach to security will need to be developed—security models that leverage insights from multiple domains such as cryptography, distributed systems, and mechanism design among others will become critical.

As evidenced by this dissertation, attacker capabilities are evolving too, creating novel challenges. One such game-changing capability also lurks on the horizon. As AI bots become indistinguishable from humans, it will be challenging to prevent attackers from using an army of them to "extract" more value than desired from our systems. Ideas of permissionless security taken from the blockchain setting may be helpful to prevent such misuse.

Another increasingly important consideration is on the robustness of our protocols. Because security models inherently try to capture real-world properties, they *will* be incomplete. Cryptographic protocols typically require assumptions to prove security. Some of these are mathematical in nature—like hardness of prime factorization or discrete log. But many are also *soft* assumptions on the state of the world—assumptions like honest majority, a trusted setup phase, or alignment of incentives. These are far more likely to be violated in practice and should be accounted for when designing modern protocols. Moreover, the assumptions made for protocol security today may not be appropriate to make in the future. Of course, this applies even to the models in this dissertation. To paraphrase a common aphorism, "all *security* models are wrong, but some are *currently* useful." As a broad point, we must think carefully about the robustness of our systems in light of inevitable failure of the assumptions we make. The hope is that doing so will uncover unexpected connections that inform our perspective on what systems security should mean.

Part III

Appendix

APPENDIX A

APPENDIX FOR CHAPTER 2

A.1 Deferred Proofs for Fairness Properties

Proof of Theorem 2.7. The proof takes inspiration from the counterexample in Example 2.6. Denote the nodes in the system by the numbers 1 to n. We show a specific environment $\mathcal{Z}$ in which no protocol can achieve receive order-fairness. Suppose that clients submit n transactions tx_1 to tx_n. Further, suppose that node 1 receives the transactions in the order $\mathsf{tx}_1, \mathsf{tx}_2, \cdots, \mathsf{tx}_n$ and any node $i \neq 1$ receives the transactions in the order $\mathsf{tx}_i, \cdots, \mathsf{tx}_n, \mathsf{tx}_1, \cdots, \mathsf{tx}_{i-1}$.

Now, it is straightforward to see that all nodes except node 2 received tx_1 before tx_2, all nodes except node 3 received tx_2 before tx_3 and so on. Finally, all nodes except node 1 received tx_n before tx_1. This means that any consensus protocol that provides order-fairness for $\gamma \leq \frac{n-1}{n}$ must order tx_1 before $\mathsf{tx}_2, \cdots,$ tx_{n-1} before tx_n, and tx_n before tx_1 which is a contradiction. □

Proofs of Theorem 2.8. The case for $\gamma < 1$ is handled by Theorem 2.7. To show the result for $\gamma = 1$, first suppose that there is a protocol Π that satisfies consistency, $(T_{\text{warmup}}, T_{\text{confirm}})$ liveness and $(1, T_{\text{warmup}})$ receive-order-fairness w.r.t all $(\mathcal{A}, \mathcal{Z})$. Suppose that the nodes in the protocol are numbered from 1 to n.

Let $\mathcal{Z}$ be the same environment considered in the proof of Theorem 2.7. We will consider adversaries that corrupt a single node. Suppose that an adversary $\mathcal{A}_1$ corrupts node 1, which claims to have received transaction tx_n before tx_1 (as opposed to the actual ordering of tx_1 before tx_n it received from the environ-

ment). Note that to all other protocol nodes, this is indistinguishable from the world where the environment itself provided node 1, the transaction tx_n before tx_1. Since in this world, Π would have to result in all honest nodes ordering tx_n before tx_1 (since all nodes received tx_n before tx_1), the same needs to hold true in $(\mathcal{A}_1, \mathcal{Z})$

In a similar spirit, we can consider other such adversaries $\mathcal{A}_i$ that corrupt node i and claim to have received tx_{i-1} before tx_i. Using the same analysis as before, we can infer than Π would also have to result in all honest nodes ordering tx_{i-1} before tx_i in $(\mathcal{A}_i, \mathcal{Z})$. But, all $(\mathcal{A}_i, \mathcal{Z})$ also cannot be distinguished since the identity of the adversarial node is unknown to other nodes.

Consequently, with respect to $(\mathcal{A}_1, \mathcal{Z})$, the protocol Π must result in all honest nodes delivering tx_1 before tx_2, $\cdots$, tx_{n-1} before tx_n, and tx_n before tx_1 which is a contradiction. □

Proof of Theorem 2.10. Consider any $(\mathcal{A}', \mathcal{Z}')$ that respects $(n, f, \Delta'_{\mathsf{int}}, \Delta'_{\mathsf{ext}})$-classical execution. Construct a $\mathcal{Z}$ that is similar to $\mathcal{Z}'$ except that between any two rounds $\mathcal{Z}$ inserts ξ rounds of "silence" i.e. it takes no action during these rounds. Note that $(\mathcal{A}', \mathcal{Z})$ also respects $(n, f, \Delta_{\mathsf{int}}, \Delta_{\mathsf{ext}})$-classical execution which would mean that Π satisfies $(\gamma, T_{\text{warmup}}, \xi)$ approximate-order-fairness.

Now, for two transactions tx_1 and tx_2 such that γ fraction nodes received tx_1 before tx_2 in $(\mathcal{A}', \mathcal{Z}')$ (and all nodes received them after time T_{warmup}), they were received more than ξ rounds apart by γ fraction nodes in $(\mathcal{A}', \mathcal{Z})$ which means tx_1 must be ordered before tx_2. Consequently, Π must result in honest nodes ordering tx_1 before tx_2 even in $(\mathcal{A}', \mathcal{Z}')$. In other words, Π will satisfy $(\gamma, T_{\text{warmup}})$ receive-order-fairness w.r.t. $(\mathcal{A}', \mathcal{Z}')$. The result follows. □

APPENDIX B

APPENDIX FOR CHAPTER 3

B.1 Deferred Proofs for Aequitas Building Blocks

Proof of Lemma 3.2. The proof is straightforward. We ignore the negligible "bad" views and let VIEW be a execution of Π_{sba} where agreement, inclusion validity, and exclusion validity are all satisfied. Suppose that there was a value c in the output agreed upon by honest nodes even though it was not in any honest node's input set. Now, to an honest node, this protocol execution is indistinguishable from the world where none of the malicious nodes had c in their input set (from $\mathcal{Z}$) either. In particular, it could have been the case that a malicious node did not receive c as input from $\mathcal{Z}$ but still proposed it as part of the protocol. Equivalently, in this world, c was in the agreed upon output in Π_{sba} even when no node was given it as input by $\mathcal{Z}$. This contradicts the exclusion validity property of Π_{sba}. □

Proof of Lemma 3.3. The proof follows in a straightforward way from the security of Π_{BBA}. Agreement and validity (both inclusion and exclusion) for Π_{sba} follow directly from the agreement and validity properties of Π_{BBA}. To see why exclusion validity holds, suppose that there is an element c that was not in any input set. This means that all honest nodes sent input 0 to the instance $\Pi_{\text{BBA}}[(\mathsf{sid}, c)]$, which implies that c cannot be in the agreed upon output set.

One crucial point worth mentioning here is that Π_{sba} forks only a polynomial number of instances of Π_{BBA} since S is of polynomial size. Consequently, all nodes still run in polynomial time. □

B.2 Deferred Proofs for the Synchronous Aequitas Protocol

Proof of Lemma 3.7. Suppose that $\mathsf{tx}_{\mathsf{first}}$ was received by some node at time r_{first}. Then, all nodes have received $\mathsf{tx}_{\mathsf{first}}$ as input by time $r_{\mathsf{first}} + \delta_{\mathsf{ext}}$. Similarly, suppose that $\mathsf{tx}_{\mathsf{last}}$ was received last by some node at time r_{last}. Then, no node has received $\mathsf{tx}_{\mathsf{first}}$ as input before time $r_{\mathsf{last}} - \delta_{\mathsf{ext}}$. Since there is no edge from v_1 to v_2, it cannot be the case that all nodes received $\mathsf{tx}_{\mathsf{first}}$ before $\mathsf{tx}_{\mathsf{last}}$. Therefore, $r_{\mathsf{last}} - r_{\mathsf{first}} \leq 2\delta_{\mathsf{ext}}$. □

B.2.1 Consistency Proof

Proof of Lemma 3.8. Suppose for contradiction, that there is a transaction tx' such that $\mathsf{tx}' \in \mathsf{TXS}_i(v)$ and $\mathsf{tx}' \notin \mathsf{TXS}_j(v')$ when tx was output by the two nodes. Since tx and tx' are in the same strongly connected component in G_i, there is a path from tx' to tx which also means that there is a path from tx' to tx in G_j when tx gets added to G_j. Since tx' is not in the same component as tx in G_j, this implies that for j to output tx, it must have output tx' before.

But tx and tx' being in the same component in G_i also implies that there is a path from tx to tx'. Consequently, j also needs to wait for tx before delivering tx'. This implies that tx and tx' are in the same strongly connected component in G_j which contradicts the assumption. The result follows. □

Proof of Theorem 3.9. Suppose that an honest node delivers a transaction tx_1 before another one tx_2. We need to show that no honest node will deliver tx_2 without delivering tx_1 first. Consider two honest nodes i and j. Let $v_{(1,i)} = \mathsf{SCC}_i(\mathsf{tx}_1)$ and $v_{(2,i)} = \mathsf{SCC}_i(\mathsf{tx}_1)$ be vertices in G_i^* when tx_1 and tx_2 were delivered. Fur-

ther, let $v_{(1,j)} = \mathsf{SCC}_j(\mathsf{tx}_1)$ and $v_{(2,j)} = \mathsf{SCC}_j(\mathsf{tx}_1)$ be vertices in G_j^* when tx_1 and tx_2 were delivered. From Lemma 3.8, we know that $v_1 = v_{(1,i)} = v_{(1,j)}$ and $v_2 = v_{(2,i)} = v_{(2,j)}$. Further, $\mathsf{TXS}_i(v_1) = \mathsf{TXS}_j(v_1)$ and $\mathsf{TXS}_i(v_2) = \mathsf{TXS}_j(v_2)$. Now, either tx_1 was delivered even before tx_2 was added to G_i, or there is an edge from v_1 to v_2 in G_i^* (which caused tx_1 to be output before) or v_1 and v_2 are incomparable.

- If tx_1 was delivered before tx_2 was added to G_i, then at least $\gamma n - f$ nodes received tx_1 before tx_2. Therefore, even if tx_2 gets added to G_j before tx_1, there will be an edge from tx_1 to tx_2 in G_j. By Lemma 3.8, tx_1 cannot be in the same SCC as tx_2 either, which implies that node j cannot deliver tx_2 first.
- If (v_1, v_2) is an edge in G_i^*, then it will also be in G_j^* when j delivers $\mathsf{TXS}(v_2)$. This means that j cannot deliver $\mathsf{TXS}(v_2)$ before it delivers $\mathsf{TXS}(v_1)$.
- If there is no edge between v_1 and v_2 in G_i^*, then node i delivers $\mathsf{TXS}(v_1)$ before because v_1 had more descendants (or because of the deterministic tie-breaker). Since j waits for $2\delta_{\mathsf{ext}}$ time, both v_1 and v_2 are present in its graph G_j^* when j outputs $\mathsf{TXS}(v_2)$, causing j to wait for a common descendant of v_1 and v_2 to be added. By waiting for another δ_{ext} rounds, any other vertex that is not a common descendant will also be in G_j^*, and the difference in the number of descendants of v_1 and v_2 will remain constant henceforth. This means that j will take the same decision as i to deliver $\mathsf{TXS}(v_1)$ before $\mathsf{TXS}(v_2)$.

The consistency result follows. □

B.2.2 Liveness Proof

Proof of Lemma 3.10. Suppose that $\mathsf{tx}' = \mathsf{tx}_0$ was in the same strongly connected component as $\mathsf{tx} = \mathsf{tx}_k$ for an honest node i. This means that there are transactions $\mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_{k-1}$ such that the edges $(\mathsf{tx}' = \mathsf{tx}_0, \mathsf{tx}_1)$, $(\mathsf{tx}_1, \mathsf{tx}_2)$, $\ldots$, $(\mathsf{tx}_{k-1}, \mathsf{tx} = \mathsf{tx}_k)$ are in the graph G_i. This means that at least one honest node received tx_0 before tx_1, at least one honest node received tx_1 before tx_2 and so on. This means that by construction of T, tx' would be in T which is a contradiction. We conclude that $\mathsf{SCC}_i(\mathsf{tx}) \neq \mathsf{SCC}_i(\mathsf{tx}')$. □

Proof of Theorem 3.11. Suppose that tx was first input by $\mathcal{Z}$ in round $r > T^{\text{fifocast}}_{\text{warmup}}$. Then, tx is received by all honest nodes as input by round $r + \delta_{\mathsf{ext}}$ and consequently added to all honest graphs G_i by round $r + 2\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}}$. Finally, tx is part of V_{finalize} for all honest nodes by round $r + 5\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}}$.

Now, consider the set T built form tx as in the weak-liveness definition. Suppose now that a transaction $\mathsf{tx}_{\mathsf{flush}}$ is input to all nodes after all transactions in T. Let r_{flush} be the round that $\mathsf{tx}_{\mathsf{flush}}$ is first input to some node. Then, $\mathsf{tx}_{\mathsf{flush}}$ is received by all nodes by round $r_{\mathsf{flush}} + \delta_{\mathsf{ext}}$ and therefore added to all honest graphs G_i by round $r_{\mathsf{flush}} + 2\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}}$. From Lemma 3.10, $v = \mathsf{SCC}_i(\mathsf{tx}) \neq \mathsf{SCC}_i(\mathsf{tx}_{\mathsf{flush}})$ for any honest i. Now, any transaction tx' that tx is incomparable was input to at least one honest node no later than tx, i.e., $\mathsf{tx}_{\mathsf{flush}}$ was received after tx' by all honest nodes. Consequently, $\mathsf{tx}_{\mathsf{flush}}$ will be a descendant of both tx and tx'. This means that node i can deliver $\mathsf{TXS}_i(\mathsf{tx})$ when $\mathsf{TXS}_i(\mathsf{tx}) \in V_{\mathsf{finalize}}$, and when $\mathsf{tx}_{\mathsf{flush}}$ gets added to its graph. Since the bound δ_{ext} could be much larger than the actual network delay, we can guarantee that both conditions are met for all honest nodes only by round $r_{\mathsf{flush}} + 5\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}} - 1$ (since

$r + 1 \leq r_{\mathsf{flush}}$ by construction).

The result follows. □

B.2.3 Fairness Proof

Proof of Theorem 3.12. The proof is straightforward. Let tx_1 and tx_2 be two transactions that all nodes receive after time $T^{\text{fifocast}}_{\text{warmup}}$. First, we note that if γn nodes receive tx_1 before tx_2, then at least $\gamma n - f$ honest ones do. This means that then there will be an edge from tx_1 to tx_2 in all honest G_i. Consequently, either tx_1 will be delivered before tx_2 by all nodes, or it will end up in the same strongly connected component as tx_2 and be delivered at the same time. In other words, tx_1 cannot be delivered at a later index than tx_2 in the LOG. □

Proof of Corollary 3.12.1. When $\delta_{\mathsf{ext}} = 1$, there are no cycles in the transaction dependencies. In other words, there are no cycles in G_i for an honest node i, and each transaction is in a strongly connected component by itself. Therefore, if at least γn nodes receive tx_1 before tx_2, there will be an edge from tx_1 to tx_2 in all honest G_i, and consequently, node i will deliver tx_1 before tx_2. □

B.2.4 Modified Synchronous Protocol

Proof of Theorem 3.13. The main component of the proof is actually to prove the consistency of Π^*. We note that liveness essentially follows directly from the construction of Π^*, while order-fairness follows in the same way as for $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ (see Section 3.3.5; Theorem 3.12 and Corollary 3.12.1). We sketch

the consistency proof below, reusing several components from the consistency proof for $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ (see Theorem 3.9).

Consistency Proof Sketch. Consider a transaction tx input to honest node i in round r. First, note that any transaction tx′ that is incomparable with tx is input to some honest node no later than round $r+\delta_{\mathsf{ext}}$, which implies i will have received it as input no later than round $r + 2\delta_{\mathsf{ext}}$. Consequently, tx′ is added to G_i by round $r + 3\delta_{\mathsf{ext}} + T^{\mathsf{fifocast}}_{\mathrm{confirm}} + T^{\mathsf{sba}}_{\mathrm{confirm}}$. In other words, any incomparable transaction is present in G_i when node i decides on the ordering for tx. We now need to show the ordering among these incomparable vertices (based on number of descendants) will be the same, resulting in all nodes taking the same ordering decision regarding tx.

Specifically, let tx′ be incomparable with tx and suppose another transaction tx* is a descendant of tx but not of tx′ in some honest G_j. We need to show that, for any honest node j, tx* is present in G_i, when i delivers tx. We note that such a tx* will be received by some honest node by round $r + 2\delta_{\mathsf{ext}}$ (which is the last round some node could have received tx′), which implies that all nodes (including node i) will receive it by round $r + 3\delta_{\mathsf{ext}}$. Consequently, tx* will be added to G_i by round $r+4\delta_{\mathsf{ext}}+T^{\mathsf{fifocast}}_{\mathrm{confirm}}+T^{\mathsf{sba}}_{\mathrm{confirm}}$. In other words, any transaction that increases the number of descendant only for one of tx and tx′ will be present in G_i by this time. Consequently, the difference in the number of descendants of tx and tx′ will be the same for all honest nodes when they deliver tx, i.e., all honest nodes will order tx and tx′ the same way.

We note that the other parts from the consistency proof for $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$ (Section 3.3.3; Theorem 3.9) now apply.

For completeness, to see why liveness holds, suppose that a transaction tx is first input in round r. Then, it is received as input by all nodes by round $r+\delta_{\mathsf{ext}}$. Since nodes deliver a transaction $4\delta_{\mathsf{ext}} + T^{\text{fifocast}}_{\text{confirm}} + T^{\text{sba}}_{\text{confirm}}$ rounds after they receive it as input, we note that all nodes will deliver tx by round $r+5\delta_{\mathsf{ext}}+T^{\text{fifocast}}_{\text{confirm}}+T^{\text{sba}}_{\text{confirm}} = r + T^{*}_{\text{confirm}}$. □

B.3 Deferred Proofs for the Asynchronous Aequitas Protocol

B.3.1 Consistency Proof

Suppose that an honest node i delivers transactions in $v_1 = \mathsf{SCC}_i(\mathsf{tx}_1)$ before $v_2 = \mathsf{SCC}_i(\mathsf{tx}_2)$. We first note that the proof for Lemma 3.8 carries over even for the asynchronous setting. This implies that for any honest node j and a transaction tx, $\mathsf{SCC}_j(\mathsf{tx}) = \mathsf{SCC}_i(\mathsf{tx}) = \mathsf{SCC}(\mathsf{tx})$. Now, one of the following three cases can arise:

1. tx_1 was delivered by i even before tx_2 was added to G_i. This means that at least $n - 2f$ logs for indices in L^{tx_2} contained tx_1 before tx_2. Consequently, for any other honest node j, even if tx_2 was added to G_j before, an edge from tx_1 to tx_2 would also be added. Since tx_1 and tx_2 are not in the same strongly connected component, this implies that j cannot deliver tx_2 before first delivering tx_1.
2. $(v_1, v_2) \in G^*_i.E$. This means, that for any honest node j, G^*_j would also have this edge. Consequently, all honest nodes will deliver transactions in

v_1 before transactions in v_2.

3. v_1 and v_2 are incomparable in G_i^*. Consequently, tx_2 was present before tx_1 in at least $f+1$ logs which implies that the node tx_2 was present in G_i when tx_1 was delivered. Now, from the description of $\Pi^{\text{async,nolead}}_{\text{Aequitas}}$, i needs to wait for a common descendant of v_1 and v_2 as well as any vertices it is incomparable with to be received and added to the graph. Let $\mathsf{Desc}_i(v_1)$ and $\mathsf{Desc}_i(v_2)$ be the descendants in G_i^* of v_1 and v_2 respectively.

   Now, let $v' \in \mathsf{Desc}_i(v_1); v' \notin \mathsf{Desc}_i(v_2)$. That is, v' is a descendant of v_1 but not of v_2. We need to show that v' is present in G_j^* for an honest j, before j delivers tx_1 or tx_2. First, we note that since v_1 and v_2 are incomparable, both are present in G_j^* before j delivers either one. This means that j also needs to wait for a common descendant v_{common} of v_1 and v_2 to be received and added to j's graph. Now, v' cannot be a descendant of v_{common} (otherwise it would also be a common descendant). Therefore, either there is an edge from v' to v_{common} in G_j^* or v' and v_{common}. In either case, j needs to wait for v' to be received and added to its graph.

   This implies that any vertex that is a descendant of exactly one of v_1 and v_2 is also present in G_j^* when j is deciding whether to output transactions in v_1 or v_2 first. Consequently, the difference in the number of descendants between v_1 and v_2 is the same as when i made its decision. In other words, j will also deliver tx_1 before tx_2.

We conclude that any honest node j will also deliver tx_1 before tx_2.

B.3.2 Liveness Proof

Suppose that a transaction tx was input to some node in the system. Eventual delivery in the external network guarantees that all nodes will eventually receive tx. Subsequently, eventual delivery in the internal network guarantees that the agreement phase for tx will eventually end resulting in all nodes adding tx to their "waiting graph." In other words, tx will eventually make its way into the graph G_i for all honest nodes i.

Consider the set T built from tx as in the weak-liveness definition. Suppose now, that a transaction tx_{flush} is input to all nodes after all transactions in T. First, we note that eventual delivery (in both the external and internal networks) guarantees that every honest graph G_i will also eventually contain tx_{flush}.

Now, we note that the proof of Lemma 3.10 also applies to the asynchronous setting. This means that tx and tx_{flush} will be in different strongly connected components in all honest G_i. Since tx_{flush} is input after tx to all nodes, it will be a descendant of tx in all honest G_i^*.

Consider any other transaction tx′ that is incomparable with tx in some honest G_i. Then, tx′ was received by at least one honest node no later than tx, i.e., tx_{flush} was input to all honest nodes after tx′. Consequently, tx_{flush} will also be a descendant of tx′ in all honest G_i.

This means that when all such tx′ get added to G_i, node i will be able to deliver tx based on which of its incomparable vertices has a larger number of descendants. Since eventual delivery guarantees that any input transaction will eventually be added to the graph, we conclude that all honest nodes will eventually deliver transaction tx.

The result follows.

B.3.3 Fairness Proof

This proof proceeds in the same way as the batch-order-fairness proof for $\Pi^{\mathsf{sync,nolead}}_{\mathsf{Aequitas}}$.

APPENDIX C

APPENDIX FOR CHAPTER 5

C.1 Standard Permissionless Blockchain Formalism

For completeness, we include the standard abstract formalism for blockchain protocols. An abstract blockchain chain is an ordered sequence of blocks where $\mathsf{chain}[i]$ is a vector of transactions contained in chain at index i. We define the requirements for an abstract blockchain next.

Definition C.1 (Abstract Blockchain Requirements)**.** Except with negligible probability over a random choice of VIEW in the support of $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$, the following need to hold:

- (T-Common-Prefix) If chain^r and chain^t are two honest chains (possibly from the same node) in rounds r and t respectively, and $r \leq t$, then $\mathsf{chain}^r[:-T] \preceq \mathsf{chain}^t$. This will allow us to confirm all except the last T blocks in chain.
- ((T, g_0, g_1)-Chain-Growth) Let chain^r and chain^t be two honest chains (possibly from the same node) in rounds r and t respectively such that $r \leq t$.
  - (Consistent length) If $t \geq r + \Delta$, $\left|\mathsf{chain}^t\right| \geq |\mathsf{chain}^r|$.
  - (Chain growth lowerbound) If $g_0 \cdot (t-r) \geq T$, then $\left|\mathsf{chain}^t\right| - |\mathsf{chain}^r| \geq \lfloor g_0 \cdot (t-r) \rfloor$.
  - (Chain growth upperbound) If $g_1 \cdot (t-r) \geq T$, then $\left|\mathsf{chain}^t\right| - |\mathsf{chain}^r| \leq \lceil g_1 \cdot (t-r) \rceil$.

- ((T, μ)-Chain-Quality) If chain is honest, then for any T consecutive blocks, at least μT blocks were mined by honest nodes.

Security of Nakamoto's protocol [203]. We recall the main properties for Nakomoto's PoW protocol. Consider Nakamoto's protocol $\Pi_{\mathsf{nak}}(p)$ with hardness parameter $p(\cdot)$. Let α be the probability that some honest nodes mines a block in one round and let ν be the expected number of adversarial blocks mined in one round. Different quantities are considered for honest and adversarial nodes since any honest chain can grow by at most one block in any round while an adversary, who can query the H sequentially can append more than one block. Then, $\alpha = 1 - (1 - p(\lambda))^{(1-\beta)qn}$ and $\nu = \beta qnp(\lambda)$ [203][1]. Define $\alpha' = \frac{\alpha}{1+\alpha\Delta}$. Define a Nakamoto-compliance predicate, denoted by $\Gamma^{p}_{\mathsf{nak}}(n, \beta, \Delta) = 1$ if for all λ, there is some $\lambda > 1$ such that $\alpha(1-2(\Delta+1)\alpha) \geq \lambda\nu$. Then, in any environment that satisfies the Nakamoto-compliance predicate, or equivalently, for a sufficiently small hardness parameter, any $T_0 = \omega(\log \kappa)$, and any constant $\varepsilon > 0$, $\Pi_{\mathsf{nak}}(p)$ satisfies T_0-consistency; (T_0, g_0, g_1)-chain-growth where $g_0 = (1 - \varepsilon)\alpha'$, $g_1 = (1 + \varepsilon)qnp$; and (T_0, μ) chain quality where $\mu = 1 - (1 + \varepsilon)\frac{\nu}{\alpha'}$. Note that all these quantities are functions of λ.

Proof simplifications. To make our proofs cleaner, we will make the following standard simplifications.

- (Simplification 1) (Mining is much slower than block propagation.)

  When the hardness parameter is sufficiently small, (e.g., when pq is small compared to $1/n$), we have $\alpha \approx q(1-\beta)np$. Therefore, by the union bound, the probability that no honest block is mined in Δ rounds is at most $(\Delta)\alpha$.

[1] We consider q queries to H per round for each node whereas [203] considers a single query.

If this is small, then with overwhelming probability, no other honest blocks are mined in the time it takes for an honestly mined block to be propagated to other nodes. This is reasonable since network propagation in Bitcoin is a few seconds, while a new block is mined roughly once every 10 minutes. For sufficiently small hardness parameter $p(\cdot)$, Pass et al. [203] showed that this happens with overwhelming probability in their proofs. We note that some prior works (eg., Prism [22]) also directly make a similar assumption.

- (Simplification 2) When the hardness parameter is sufficiently small, $\frac{\nu}{\alpha'}$ is well approximated by $\frac{\beta}{1-\beta}$, and therefore $\mu = 1-(1+\varepsilon)\frac{\nu}{\alpha'}$ is well approximated by $1-(1+\varepsilon)\frac{\beta}{1-\beta}$. Then, for any $\beta < \frac{1}{3}$, there will exist some ε_0 such that Π_{nak} satisfies chain quality with $\mu > \frac{1}{2}$. Specifically, for any $\epsilon > 0$ and $\beta = 1/3 - \epsilon$, let $0 < \varepsilon_0 < 1 - \frac{2+3\epsilon}{2-\epsilon}$. Then, Π_{nak} satisfies (T_0, μ_0)-chain quality where $T_0 = \omega(\log \lambda)$ and $\mu_0 > \frac{1}{2}$. In other words, in any T_0 consecutive blocks, the majority of blocks are mined by honest miners. In our proofs, we will directly use majority-chain-quality when $\beta < \frac{1}{3}$.

C.2 Player Replaceability Proof

We directly prove a more general version of the player replaceability lemma.

Lemma C.2 (Player Replaceability (General Form)). *Consider an $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \cdot, \cdot)$-permissionless execution, an order-fairness parameter γ, an admissible $\Upsilon = (d, d\beta', \gamma)$ for some $0 \le \beta' \le 1$. Suppose that Π is a protocol as follows: For any node N that is honest in round r, N outputs $\mathcal{L}_N^r = [\mathsf{List}_1^r, \ldots, \mathsf{List}_d^r]$ to $\mathcal{Z}$ that satisfies the following:*

1. *If N is also honest in round $r-1$, then $\mathcal{L}_N^{r-1} \preccurlyeq \mathcal{L}_N^r$.*

2. *For any pair of transactions* $(\mathsf{tx}, \mathsf{tx}')$ *that satisfies the antecedent of the (permissionless)* γ*-order-fairness definition (Definition 5.3), if either* tx *or* tx′ *is present in all* d *lists in* $\mathcal{L}_N^r$*, then* tx *will be* γ*-globally preferred to* tx′ *w.r.t.* $\Lambda = (\Upsilon, \mathcal{L}_N^r)$*. In other words, the edge* $(\mathsf{tx}, \mathsf{tx}')$ *will be in the dependency graph* $\mathsf{Graph}(\mathcal{L})$*.*

Suppose further that an algorithm $F(\cdot)$ *satisfies the streaming strong (resp. weak) fair-order finalization problem for* Υ *such that the* linearize *function of* Π *is defined to be* $F(\mathcal{L}_N^r)$*. Then,* Π *satisfies* γ*-receive-order-fairness (resp.* γ-batch-order-fairness*) w.r.t.* $(\mathcal{A}, \mathcal{Z})$*.*

Proof. For any pair $(\mathsf{tx}, \mathsf{tx}')$ that satisfies the antecedent of the (permissionless) γ-order-fairness definition, we are given that Π is such that tx will be γ-globally preferred to tx′ in the d lists that represent the virtual parties. Now, since $F(\cdot)$ satisfies the strong (resp. weak) fairness of ordering property, the final output ledger of an honest node in the given round, will contain tx before (resp. no later than) tx′. □

C.3 Protocol Security Proofs

We first begin with the proof of the helper Lemma 5.9.

Proof of Lemma 5.9. Let $(\mathsf{tx}_1, \mathsf{tx}_2, \ldots, \mathsf{tx}_l)$ be a Condorcet cycle in $\mathcal{L}$ and suppose that tx and tx′ are in this cycle. Let round r_{first} be the first round that any one of $\{\mathsf{tx}_1, \ldots, \mathsf{tx}_l\}$ was received by an honest node. We can assume tx_l was first received by an honest node at round r_{first} without loss of generality by rotating the tuple cyclically. This means that any honest node that spawned no later than

$r_{\text{first}} + 1$, received tx_l either in round r_{first} or $r_{\text{first}} + 1$, and any honest node that spawned in round $r' > r_{\text{first}} + 1$ received tx_l in round r'.

Now, suppose that tx_{l-1} was first received by an honest node at round r_{l-1} and suppose for the sake of contradiction that $r_{l-1} \geq r_{\text{first}} + 1$. This means that any honest node that spawned no later than r_{l-1} had already received tx_l (i.e., either tx_l before tx_{l-1} or tx_l and tx_{l-1} in the same round) and any node that spawned after received both tx and tx′ at the same time. However, since we have $\text{tx}_l \lhd \text{tx}_{l-1}$, this contradicts the dependency-faithfulness of Π since there needs to be some honest node that received tx before tx′. Furthermore, since no transaction in $\{\text{tx}_1, \ldots, \text{tx}_l\}$ was received before r_{first}, this means that tx_{l-1} was first received by an honest node in round r_{first}.

Continuing in the same fashion, the same argument shows that any transaction in $\{\text{tx}_1, \ldots, \text{tx}_l\}$, including tx and tx′, was first received by an honest node in round r_{first}.

Note that also applies to the case where some of the transactions are adversarial (i.e. first seen by the network as part of a block rather than transaction gossip). Recall that the ledger synchrony assumption of our network ensures that transactions are provided as input to all honest nodes if some honest node outputs them to the environment. □

C.3.1 Security Proof for Π_{mod} (Theorem 5.6)

We start by showing that Π_{mod} is dependency-complete which will allow us to conclude order-fairness.

Lemma C.3. *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$. *Then,* $\Pi^{\lambda,T}_{\text{mod}}(p)$ *is* γ*-dependency-complete for* $(n, \beta, \Delta, \tau, R)$ *environments.*

Proof. Consider any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$-permissionless execution. Consider a pair of transactions $(\mathsf{tx}, \mathsf{tx}')$ that satisfies the antecedent of the order-fairness definition. For a given chain, suppose that $\mathsf{chain}[b]$ is the first block that contains either tx or tx′ and that $|\mathsf{chain}| \geq b + 2T - 1$. Note that all blocks in $\mathsf{chain}[b, b + T - 1]$, and consequently all modular chains, will include either tx or tx′. Now, since $\gamma - (\beta + \tau) > \frac{2}{3}$ and $\mathcal{A}$ is (τ, R)-respawning, there are more than $\frac{2n}{3}$ nodes that received tx before tx′ and are honest until all modular chains contain either tx or tx′ (i.e., until $\mathsf{chain}[b + T - 1]$ has been confirmed). Let H be the set of these nodes. Now, from the chain-quality of $\Pi_{\text{nak}}(p)$, in the range $\mathsf{chain}[b, b + T - 1]$, more than $\frac{T}{2}$ of these blocks are mined by the nodes in H. In other words, more than half of the blocks in $\mathsf{chain}[b, b + T - 1]$ will contain tx before tx′. Since these blocks have been confirmed in chain, more than half of the lists in $\mathcal{L} = \mathsf{modularize}(\mathsf{chain}[: -T], T)$ will contain tx before tx′. Since $\Upsilon = (T, \lceil \frac{T}{2} \rceil - 1, 1)$ is admissible, tx is globally preferred to tx′ w.r.t. $(\Upsilon, \mathcal{L})$, i.e. $\mathsf{tx}' \lhd^{(\Upsilon, \mathcal{L})} \mathsf{tx}$.

We conclude that $\Pi^{\lambda,T}_{\text{mod}}$ is dependency-complete. □

Theorem C.4 (Order-Fairness of Π_{mod})**.** *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$. *Then,* $\Pi^{\lambda,T}_{\text{mod}}(p)$ *satisfies* γ-batch-order-fairness.

Proof. This is a straightforward combination of Fact 4.7, Lemma 5.5, and Lemma C.3. Note that for the simplest case of $\gamma = 1$, we require $\beta + \tau < 1/3$. □

Next, we show that Π_{mod} is dependency-faithful in the following lemma.

Lemma C.5. *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta, \gamma, T, \tau, R) = 1$. *Then,* $\Pi^{\lambda,T}_{\text{mod}}(p)$ *is dependency-faithful for* $(n, \beta, \Delta, \tau, R)$ *environments.*

Proof. Consider any $(\mathcal{A}, \mathcal{Z})$ that satisfies $(n, \beta, \Delta, \tau, R)$-permissionless execution. Suppose that a node N is honest in round r, holds chain, and outputs $\mathcal{L} = \mathsf{modularize}(\mathsf{chain}[: -T], T)$ to $\mathcal{Z}$. Note that $\mathcal{L}$ only contains the confirmed part of each modular chain. Let $\Upsilon = (T, \lceil \frac{T}{2} \rceil - 1, 1)$ and $\Lambda = (\Upsilon, \mathcal{L})$. Suppose that $\mathsf{tx}' \lhd^{\Lambda} \mathsf{tx}$, and let $\mathsf{chain}[b]$ be the first block that contains either tx or tx′. Then, the range $\mathsf{chain}[b, b + T - 1]$ will determine the ordering between tx and tx′ in each of the T modular chains. Since $\mathsf{tx}' \lhd^{\Lambda} \mathsf{tx}$, more than $\frac{T}{2}$ of these blocks order tx before tx′. But, since $\beta < \frac{1}{3}$ (from $\gamma - (\beta + \tau) > \frac{2}{3}$), the chain-quality property of $\Pi_{\text{nak}}(p)$ implies that less than half of the blocks in the range $\mathsf{chain}[b, b+T-1]$ are mined by an adversary. This means that there is at least 1 block that contained tx before tx′ which was mined by an honest node. Therefore, some honest node received tx before tx′ from $\mathcal{Z}$.

We conclude that $\Pi^{\lambda,T}_{\text{mod}}$ is dependency-faithful. □

We can now use the chain-growth of Π_{nak} to bound the liveness for Π_{mod} in the following theorem.

Theorem C.6 (Liveness of Π_{mod})**.** *Consider parameters that satisfy* $\Gamma^{p}_{\text{mod}}(n, \beta, \Delta = 1, \gamma, T, \tau, R) = 1$, *and suppose that* $\Pi_{\text{nak}}(p)$ *satisfies* (T, g_0, g_1)*-chain-growth. Then* $\Pi^{\lambda,T}_{\text{mod}}(p)$ *satisfies* $(5T + 2)/(2g_0)$*-liveness.*

Proof. Consider any transaction tx that is first input in round r. By the time more than $\frac{5T}{2}$ blocks are mined, all semantic chains will contain tx in the confirmed part of the chain as well as any transactions that are in the same cycle as tx.

Furthermore, any transaction that is first received by an honest node at $r+2$ will be a descendant of tx (as well as any other tranasction without an edge to/from tx) in the dependency graph of an honest node. Note that such a transaction will also be in the confirmed part of the chain. Consequently, by this time $\mathsf{Aequitas}(\cdot)$ will have delivered tx.

Now, we note that for a round r' such that $r' - r \geq \frac{5T+2}{2g_0}$, we have $g_0(r'-r) \geq \frac{5T}{2} + 1$, which implies that tx will be output by round r' by all honest nodes. Consequently, Π_{mod} satisfies $\frac{5T+2}{2g_0}$-liveness. □

This allows us to conclude the main security result for Π_{mod} (i.e., Theorem 5.6).

Remark 20 ($\beta < \frac{1}{2}$ using Fruitchains)**.** We can handle minority corruption by using Fruitchains [204] as the underlying consensus protocol. Fruitchains achieves optimal mining-fairness, which means that for any $\beta < \frac{1}{2}$, and a sufficiently long period T, it has $(T, \mu_0 > \frac{1}{2})$-chain-quality. Consequently, by using Fruitchains, our construction can tolerate $\beta+\tau < \frac{1}{2}$ corruption and still achieve order-fairness (for $\gamma = 1$). We provide a brief sketch of the protocol in Section 5.3.3. Since we require only the appropriate cross-chain-quality parameters, the security proofs for such as protocol will almost exactly mirror the ones for Π_{mod}.

Remark 21 (Other longest-chain protocols)**.** Note that since our proofs only utilize the abstract blockchain properties of the underlying PoW chain, our single-chain construction can also be layered on top of other longest-chain protocols that satisfy the chain-quality property (e.g., Proof-of-Stake protocols like [164]).

C.3.2 Security Proof for Π_{multi} (Theorem 5.7)

We start by showing that cross-chain-quality implies both dependency-completeness and dependency-faithfulness for Π_{multi}.

Lemma C.7. *Consider $(n, \beta, \Delta = 1, \tau, R)$, and d, and define $\Pi^{\lambda,g}_{\text{multi}}(p)$ as before. Suppose that for all $(\mathcal{A}, \mathcal{Z})$ that support $(n, \beta, \Delta = 1, \tau, R)$-permissionless-execution, $\Pi^{\lambda,d}_{\text{multi}}(p)$ satisfies cross-chain-quality (except for a negligible number of views), then it is also 1-dependency-complete and dependency-faithful.*

Proof.

1. (Dependency-Completeness) Suppose that $\Pi^{\kappa,g}_{\text{multi}}(p)$ satisfies cross-chain quality, and suppose for contradiction, that it is not dependency complete. Therefore, there is a pair of transactions $(\mathsf{tx}_1, \mathsf{tx}_2)$ that satisfies the antecedent of γ-order-fairness but an honest node outputs $\mathcal{L}$ with $\mathsf{tx}_2 \lhd^{\Lambda} \mathsf{tx}_1$ where $\Lambda = (\Upsilon, \mathcal{L})$ and $\Upsilon = (g, \lceil \frac{g}{2} \rceil - 1, 1)$.

   Now, for each chain, we consider 2 possibilities for the placement of the two transactions: either tx_1 is included before tx_2 or tx_2 is included before tx_1 (The case of both included at the same place is not relevant for our purpose). Furthermore, there are 4 possibilities for mining of the blocks containing $\mathsf{tx}_1, \mathsf{tx}_2$: both honest, both adversarial, only tx_1 honest, only tx_2 honest. We use the following notation to denote these 8 combinations: (H_1, A_2) represents that tx_1 was included before tx_2 and the block containing tx_1 was mined by an honest node while the block containing tx_2 was mined by an adversarial node. Let $\mathsf{num}(H_1, A_2)$ be the number of chains that have this combination.

Since, $\mathsf{tx}_2 \lhd^{\Lambda} \mathsf{tx}_1$, more than $\frac{g}{2}$ should have included tx_2 before tx_1. Therefore,

$$\mathsf{num}(H_2, H_1) + \mathsf{num}(H_2, A_1) + \mathsf{num}(A_2, H_1) + \mathsf{num}(A_2, A_1) > \frac{g}{2}$$

But since all honest nodes received tx_1 before tx_2 during the initial propagation, and $\Delta = 1$, any honest node that spawns later will receive tx_1 as soon as it spawns and therefore cannot have received tx_2 before tx_1. This means that, $\mathsf{num}(H_2, H_1)$ and $\mathsf{num}(H_2, A_1)$ will both be zero since no honest node will include tx_2 before tx_1. Therefore,

$$\mathsf{num}(A_2, H_1) + \mathsf{num}(A_2, A_1) > \frac{g}{2}$$

But this contradicts cross-chain-quality since the adversary was able to mined tx_2 in more than half of the chains. This completes the proof.

2. (Dependency-Faithfulness) Suppose that $\Pi^{\kappa,g}_{\mathsf{multi}}(p)$ satisfies cross-chain quality, and suppose for contradiction, that it is not dependency-faithful. Therefore, there is an honest node that outputs $\mathcal{L}$ with $\mathsf{tx}_2 \lhd^{\Lambda} \mathsf{tx}_1$ where $\Lambda = ((g, \lceil \frac{g}{2} \rceil - 1, 1), \mathcal{L})$ but there is no honest node that received tx_1 before tx_2.

   Now, using the same notation from the dependency-completeness argument, we have

$$\mathsf{num}(H_1, H_2) + \mathsf{num}(H_1, A_2) + \mathsf{num}(A_1, H_2) + \mathsf{num}(A_1, A_2) > \frac{g}{2}$$

   Since there is no honest node that received tx_1 before tx_2 (including nodes that spawn later since $\Delta = 1$), both $\mathsf{num}(H_1, H_2)$ + $\mathsf{num}(H_1, A_2)$. Therefore,

$\mathsf{num}(A_1, H_2) + \mathsf{num}(A_1, A_2) > \frac{g}{2}$, which once again will contradict cross-chain-quality for tx_1.

□

We reserve Section C.4 for the proof that Π_{multi} satisfies the cross-chain-quality property. Equipped with this, we can conclude the main security result for Π_{multi}.

C.4 Cross-chain-quality proof for Π_{multi}

We restate the cross-chain-quality property below. We will show that it holds, except for a negligible number of views.

Property 5.10 (Cross-Chain-Quality)**.** For a given VIEW and tx, let $\mathcal{B}_{\text{tx}}$ be the set of blocks that contain tx for the first time in the confirmed part (i.e. $\mathsf{chain}_j[: -T]$) of each of the d chains. Then, more than half of the blocks in $\mathcal{B}_{\text{tx}}$ were mined by honest nodes.

Notation. Before proceeding with the proof, we start by introducing some useful notation from [22]. Let $f = npq$ denote the mining rate (as a function of λ) per round for one chain. Typically, in practice, the hardness parameter p is set as $p = \Theta(\frac{1}{\Delta qn})$ which would make $f = \Theta(1)$ in λ since we are using $\Delta = 1$. This will make our analysis cleaner. Recall that d is the number of parallel chains and β is the adversarial corruption threshold. We use the following random variables in our proofs. Note that the probabilities are all taken

over VIEW $\leftarrow_{\$} \mathsf{EXEC}^{\Pi_{\text{multi}}}(\mathcal{A}, \mathcal{Z}, \lambda)$ although for succinctness, we do not explicitly write this. For a chain i, and round r, we define:

- $X_i[r] = \begin{cases} 0, \text{ if no honest node mines a block in round } r \\ 1, \text{ if at least one honest node mines a block in round } r \end{cases}$
- $X_i[r_1, r_2] = \sum_{r=r_1}^{r_2} X_i[r]$.
- $Y_i[r] = \begin{cases} 1, \text{ if exactly one honest node mines a block in round } r \\ 0, \text{ otherwise} \end{cases}$
- $Y_i[r_1, r_2] = \sum_{r=r_1}^{r_2} Y_i[r]$.
- $Z_i[r]$ is the number of blocks mined by an adversarial node in round r.
- $Z_i[r_1, r_2] = \sum_{r=r_1}^{r_2} Z_i[r]$.
- $C[:\ell]$ is the prefix of the longest chain C until the level ℓ.
- $C[\ell]$ is the block at level ℓ of the chain C.

For an event Q, use $\mathbf{1}_Q$ to denote the indicator variable for when Q holds. That is, $\mathbf{1}_Q = 1$ when Q is true and 0 otherwise. We use the notation $a_+ = \max\{0, a\}$.

Following [22], $Z_i[r]$ follows a Poisson distribution, while $X_i[r]$ follows a Bernoulli distribution. Specifically, we have,

- $Z_i[r] \sim \text{Poisson}(\beta f)$
- $X_i[r] \sim \text{Bernoulli}(1 - e^{-(1-\beta)f})$

C.4.1 Proof structure.

In order to prove the cross-chain-quality property for Π_{multi}, abstractly we will construct an event $V_i[r]$ for each chain i at round r, which will guarantee that any transaction first seen in round r will be contained in a block mined by an honest node. Now, the probability in an arbitrary chain i at any round r, that $V_i[r]$ happens (as a function of β and f) is larger than 0.5, then we can set the number of chains $d = \Theta(\lambda)$, and use the Chernoff bound to argue that the probability that any transaction is mined in more than half the parallel chains by the adversary is negligible in λ. We elaborate on the main proof parts below.

Constructing $V_i[r]$. We start by defining the following random processes for all chains i and rounds $r > 0$.

$$S_i[r] = (S_i[r-1] + Z_i[r] - X_i[r])_+$$
$$L_i[r,t] = X_i[r,t] - Z_i[r,t].$$

We also set $S_i[0] = 0$. Now, if the adversary never reveals the blocks it mines, $S_i[r]$ captures the difference between the length of longest private adversarial chain i at round r, and the longest public chain held by honest nodes. The random process $L_i[r,t]$ represents the difference between the number of honest blocks and adversarial blocks mined in chain i from round r to round t.

Now, we define $V_i[r]$ for chain i at round $r > 0$ as the event that all of the following holds:

1. $S_i[r-1] = 0$

2. There exists a round $t' > r$ such that

   - $Z_i[r, t'] = 0$. In other words, the adversary was not able to mine a block from round r to t'.
   - $X_i[r, t' - 1] = 0$. In other words, honest nodes were not able to mine any block from r until $t' - 1$.
   - $Y_i[t'] = 1$. In other words, exactly one honest node mined a block in round t'.
   - $L_i[r + 1, t] \geq 1 \quad \forall t > t'$. In other words, the number of honest blocks mined in chain i from round $r + 1$ until round t is greater than the corresponding number of adversarial blocks in the same interval, for all $t > t'$.

We will call the first event $V_i^1[r]$ and the second event $V_i^2[r]$. We evaluate their probabilities separately since they are independent.

Honest block when $V_i[r]$ holds. In Section C.4.2, we show that whenever $V_i[r]$ holds, any transaction tx that is first revealed (to honest nodes) at round r, in chain i, would be contained in a block mined by honest miner. Intuitively, there are two ways a transaction can be propagated:

- *Honestly propagated transaction*: A transaction that is input to all nodes (honest and adversarial) within $\Delta = 1$ rounds.
- *Adversarially propagated transaction*: A transaction that is created by the adversary and is first seen by an honest node only through the mining of a PoW block by the adversary. Here, the adversary knows the transac-

tion earlier than honest nodes and can get a head-start to mine it into the chains.

Our proof of the result holds regardless of how the transaction is propagated.

Probability of $V_i[r]$. In Section C.4.3, we compute the probability, as a function of β and f, that the event $V_i[r]$ occurs for a given round r. Denote this by $\phi(\beta, f)$. Next, we find the threshold β_{thresh} such that for $\beta < \beta_{\text{thresh}}$, we have $\phi(\beta, f) > 0.5$. This means that for a transaction first revealed in round r, the probability for each chain that tx is contained in a block mined by an honest node is more than 0.5.

Achieving cross-chain-quality. We can now leverage the Chernoff bound to get a lower bound on the number of chains g, so that the probability that the adversary mined tx into more than half of the parallel chains (or equivalently, cross-chain-quality is not satisfied) is negligible in the security parameter. We show this in Section C.4.4.

C.4.2 Honest block under $V_i[r]$

In this section, we will show that $V_i[r]$ guarantees that any transaction (honest or adversarial) first input to honest nodes at round r will be contained in an honest block in chain i.

For this, we need to show that the following hold, except with negligible prob-

ability.

- At round r, the adversary does not have a longer private adversarial chain for chain i than the longest honest chain (Lemma C.8).
- The first block to be mined in chain i after round r is an honest block and this honest block persists in the longest chain for all future rounds (Lemma C.9).

Lemma C.8. *Consider a round when $S_i[r] = 0$. The adversary cannot posses a longer chain than the chain held by the honest users at round r.*

Proof. The proof proceeds by contradiction. Suppose r was a round with $S_i[r] = 0$ and the adversary possessed a longer chain C_i^a. Let the level of the tip of the longest chain held by the honest users, C_i^h, at round r be ℓ_{tip}. Let,

- $\ell_{common} = \max\{\ell : C_i^a[:\ell] = C_i^h[:\ell]\}$ be the last common ancestor,
- $\ell' = \max\{\ell \leq \ell_{common}$ such that $C_i^h[\ell]$ is honest block$\}$ and r' be the round when it was mined.

Observe that for each of the blocks in the chain C_i^h from level $\ell_{common}+1$ to level ℓ_{tip}, there is a corresponding adversarial block in the adversarial chain C_i^a. Also, if $\ell' < \ell_{common}$, then all the blocks from levels $\ell'+1$ to ℓ_{common} in the chain C_i^h are adversarial blocks. Thus, from $r'+1$ to round r, the number of adversarial blocks mined is greater than the number of rounds where at least one honest block was mined. That is,

$$Z_i[r'+1, r] \geq X_i[r'+1, r] + 1$$

However, we also have $S_i[r'] \geq 0 = S_i[r]$, and so,

$$\begin{aligned} 0 &\geq S_i[r] - S_i[r'] \\ &\geq \sum_{s=r'+1}^{r} Z_i[s] - X_i[s] \\ &= Z_i[r'+1, r] - X_i[r'+1, r] \end{aligned}$$

i.e. $Z_i[r'+1, r] \leq X_i[r'+1, r]$ which is a contradiction. □

Lemma C.9. *Suppose $V_i[r]$ happens at round r. Then, the first block B mined at round $t' > r$ is honest and will persist in the longest chain.*

Proof. First, observe that since $Z_i[r, t'] = 0$, we have $S_i[r-1] = S_i[r] = \cdots = S_i[t'] = 0$. Now, by Lemma C.8, the adversary doesn't possess a longer chain in the rounds $r-1, r, \cdots, t'$. Also, $Z_i[r, t'] = 0$ implies that no new adversarial block was mined in in the rounds $r-1, r, \cdots, t'$ and exactly one honest node mines a block in round t'. So, the chain i held by honest users at round t' is longer than any adversarial chain and thus, contains the block B.

Next, by contradiction, let $r^* > t'$ be the earliest time when the honest users hold a chain that contains a different block at the same level where B was. Thus, at round r^*, there exists an adversarial chain that is at least as long the honest chain containing the block B. Let round r' be as defined in Lemma C.8. Then, reasoning as in Lemma C.8, from round $r'+1$ to round r^*, the number of adversarial blocks mined is greater than or equal to the number of rounds where at least one honest block is mined. That is,

$$Z_i[r'+1, r^*] \geq X_i[r'+1, r^*]$$

Also, we have

$$S_i[r'] \geq 0 = S_i[r]$$
$$\implies Z_i[r'+1, r] \leq X_i[r'+1, r]$$

This implies that $Z_i[r+1, r^*] \geq X_i[r+1, r^*]$. However, as event $V_i[r]$ occurs at round r, so, we have

$$L_i[r+1, r^*] \geq 1$$
$$X_i[r+1, r^*] - Z_i[r+1, r^*] \geq 1$$
$$X_i[r+1, r^*] > Z_i[r+1, r^*]$$

which leads to contradiction. □

C.4.3 Computation of $\phi(f, \beta)$

In this section, we compute the probability $\phi(f, \beta)$, of the event $V_i[r]$. For this, we will prove the following lemma.

Lemma C.10.

$$\lim_{f \to 0^+} \phi(f, \beta) = \frac{(1-2\beta)^2}{1-\beta}.$$

We split this computation into two parts. We compute the probability of the event $V_i^1[r]$ in Section C.4.3.1, and the probability of $V_i^2[r]$ given $V_i^1[r]$ in Section C.4.3.2. Finally, we can combine the two probabilities to compute $\phi(f, \beta)$.

C.4.3.1 Probability of $V_i^1[r]$

Recall that $V_i^1[r]$ is the event that $S_i[r-1] = 0$. To compute the probability of this event, we will compute the stationary distribution of the discrete time Markov chain $\{S_i[r]\}_{r\geq 0}$.

First, for a chain i, and round r, we define the following random variable:

$$S_i[r] = (S_i[r-1] + Z_i[r] - X_i[r])_+$$
$$S_i[r] = 0$$

Observe that due to the memory-less property of both honest and adversarial mining (since mining is through a random oracle), for all $r > 0$, we have,

$$P(S_i[r] = s \mid S_i[r-1] = s_{r-1}, \cdots, S_i[0] = 0) = P(S_i[r] = s \mid S_i[r-1] = s_{r-1})$$

Therefore, the process $\{S_i[r]\}_{r\geq 0}$ gives rise to a discrete time Markov chain (DTMC). We now show that $\{S_i[r]\}_{r\geq 0}$ is positive recurrent in the following lemma.

Lemma C.11. *If $\mathbb{E}[Z_i[r]] < \mathbb{E}[Y_i[r]]$, then, $\{S_i[r]\}_{r\geq 0}$ is positive recurrent.*

Proof. First, notice from the definition of $S_i[r]$ that

$$\begin{aligned}(S_i[r+1])^2 &\leq (S_i[r] + Z_i[r+1] - Y_i[r+1])^2 \\ &= (Z_i[r+1] - Y_i[r+1])^2 + (S_i[r])^2 + 2S_i[r](Z_i[r+1] - Y_i[r+1])\end{aligned}$$

Now, for each chain i, define a drift function as:

$$\mathfrak{d}S_i[r] = \frac{1}{2}(S_i[r+1])^2 - \frac{1}{2}(S_i[r])^2$$

Then,

$$\begin{aligned}\mathfrak{d}S_i[r] &\leq \frac{1}{2}(Z_i[r+1] - Y_i[r+1])^2 + S_i[r]\left(Z_i[r+1] - X_i[r+1]\right)\\ \mathbb{E}\left[\mathfrak{d}S_i[r] \mid S_i[r]\right] &\leq \left(\frac{1}{2}\mathbb{E}\left[(Z_i[r+1] - X_i[r+1])^2 \mid S_i[r]\right]\right)\\ &\quad + \left(S_i[r]\mathbb{E}\left[(Z_i[r+1] - X_i[r+1]) \mid S_i[r]\right)\right]\end{aligned}$$

Recall that $Z_i[r+1] \sim \text{Poisson}(\beta f)$ and $X_i[r+1] \sim \text{Bernoulli}(1-e^{-(1-\beta)f})$ and so, $Z_i[r+1]$ and $X_i[r+1]$ have bounded second moments. Therefore, the first term in the expression before can be bounded as $\frac{1}{2}\mathbb{E}\left[(Z_i[r+1] - X_i[r+1])^2 \mid S_i[r]\right] \leq B$, where $B = B(f, \beta)$ depends on f and β. Thus,

$$\begin{aligned}\mathbb{E}\left[\mathfrak{d}S_i[r] \mid S_i[r]\right] &\leq B + S_i[r]\mathbb{E}\left[(Z_i[r+1] - X_i[r+1]) \mid S_i[r]\right]\\ &= B + S_i[r]\left(\mathbb{E}[Z_i[r+1]] - \mathbb{E}[X_i[r+1]]\right)\end{aligned}$$

However, $\mathbb{E}[Z_i[r+1]] - \mathbb{E}[X_i[r+1]] < -\epsilon$, for some $\epsilon > 0$. Therefore, we have

$$\mathbb{E}\left[\mathfrak{d}S_i[r] \mid S_i[r]\right] \leq B - \epsilon S_i[r]$$

To summarize,

- $\mathbb{E}\left[\mathfrak{d}S_i[r] \mid S_i[r]\right] \leq -\epsilon^*$ for some $\epsilon^* > 0$, if $S_i[r] > \frac{B}{\epsilon}$
- $\mathbb{E}\left[\mathfrak{d}S_i[r] \mid S_i[r]\right] \leq B$ otherwise.

Thus, by Foster's Theorem [115], we conclude that $\{S_i[r]\}_{r\geq 0}$ is positive recurrent. □

Now, $\{S_i[r]\}_{r\geq 0}$ is irreducible since it is possible to get from any state to any other state (either by honest block mining or by adversarial block mining). Furthermore, it is aperiodic since it has self-loops, and therefore a 1-step transition

can retain the earlier state. Therefore the stationary distribution exists. If $\boldsymbol{\pi}$ is the stationary distribution of $\{S_i[r]\}_{r\geq 0}$, then, by ergodic theorem [147], we have

$$\lim_{r\to\infty} P(S_i[r]=0) = \pi_0$$

where π_0 denotes the stationary distribution probability at state 0.

We will now calculate the probability of the stationary distribution π_0 under a general f. For this, we will write the transition probability matrix for the DTMC $\{S_i[r]\}_{r\geq 0}$. Define $\mathtt{p}_X^0 = P(X_i[r]=0) = e^{-(1-\beta)f}$, $\mathtt{p}_X^1 = P(X_i[r]=1) = 1-e^{-(1-\beta)f}$, and $\mathtt{p}_Z^k = P(Z_i[r]=k) = \frac{(\beta f)^k e^{-\beta f}}{k!}$. Then, the transition probability matrix is given by

$$\begin{pmatrix}
\mathtt{p}_Z^0\mathtt{p}_X^0 + \mathtt{p}_Z^1\mathtt{p}_X^1 & \mathtt{p}_Z^1\mathtt{p}_X^0 + \mathtt{p}_Z^2\mathtt{p}_X^1 & \mathtt{p}_Z^2\mathtt{p}_X^0 + \mathtt{p}_Z^3\mathtt{p}_X^1 & \mathtt{p}_Z^3\mathtt{p}_X^0 + \mathtt{p}_Z^4\mathtt{p}_X^1 & \cdots \\
\mathtt{p}_Z^0\mathtt{p}_X^1 & \mathtt{p}_Z^0\mathtt{p}_X^0 + \mathtt{p}_Z^1\mathtt{p}_X^1 & \mathtt{p}_Z^1\mathtt{p}_X^0 + \mathtt{p}_Z^2\mathtt{p}_X^1 & \mathtt{p}_Z^2\mathtt{p}_X^0 + \mathtt{p}_Z^3\mathtt{p}_X^1 & \cdots \\
0 & \mathtt{p}_Z^0\mathtt{p}_X^1 & \mathtt{p}_Z^0\mathtt{p}_X^0 + \mathtt{p}_Z^1\mathtt{p}_X^1 & \mathtt{p}_Z^1\mathtt{p}_X^0 + \mathtt{p}_Z^2\mathtt{p}_X^1 & \cdots \\
0 & 0 & \mathtt{p}_Z^0\mathtt{p}_X^1 & \mathtt{p}_Z^0\mathtt{p}_X^0 + \mathtt{p}_Z^1\mathtt{p}_X^1 & \cdots \\
0 & 0 & 0 & \mathtt{p}_Z^0\mathtt{p}_X^1 & \cdots \\
0 & 0 & 0 & 0 & \cdots \\
\vdots & \vdots & \vdots & \vdots & \ddots
\end{pmatrix}$$

The important observation is that this matrix is of the form

$$\begin{pmatrix} a & b_2 & b_3 & \cdots \\ b_0 & b_1 & b_2 & \cdots \\ 0 & b_0 & b_1 & \cdots \\ 0 & 0 & b_0 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix}$$

with $a = b_0 + b_1$. Therefore, we get the following relationship between the stationary probabilities

$$\begin{aligned} \pi_0 &= a\pi_0 + b_0\pi_1 \\ \pi_j &= \sum_{k=0}^{j+1} b_{i+1-k}\pi_k \qquad\qquad (j > 1) \end{aligned}$$

Rearranging the terms of the second equation, we get

$$\begin{aligned} (1 - b_1)\pi_j &= b_0\pi_{j+1} + \sum_{k=0}^{j-1} b_{j+1-k}\pi_k \\ \therefore (1 - b_1)\pi_j z^{-j} &= b_0\pi_{j+1}z^{-j} + \sum_{k=0}^{j-1} b_{j+1-k}\pi_k z^{-j} \\ \therefore (1 - b_1)\sum_{j\geq 1}\pi_j z^{-j} &= b_0\sum_{j\geq 1}\pi_{j+1}z^{-j} + \sum_{j\geq 1}\sum_{k=0}^{j-1} b_{j+1-k}\pi_k z^{-j} \end{aligned}$$

Notice that the summation $\sum_{j\geq 1}\sum_{k=0}^{j-1}$ can be written in terms of $\sum_{k\geq 0}\sum_{j\geq k+1}$. Now, define the unilateral z-transforms $\mathfrak{P}(z) = \sum_{j\geq 0}\pi_j z^{-j}$ and $\mathfrak{G}(z) =$

$\sum_{j\geq 0} b_j z^{-j}$. See [147] for a primer on the z-transform. Then,

$$\begin{aligned}(1-b_1)(\mathfrak{P}(z)-\pi_0) &= b_0 z\left(\mathfrak{P}(z)-\pi_0-\frac{\pi_1}{z}\right)+\sum_{k\geq 0}\sum_{j\geq k+1} b_{j+1-k}\pi_k z^{-j} \\ &= b_0 z\left(\mathfrak{P}(z)-\pi_0-\frac{\pi_1}{z}\right)+z\sum_{k\geq 0}\pi_k z^{-k}\sum_{j\geq k+1} b_{j+1-k}z^{-(j+1-k)} \\ &= b_0 z\left(\mathfrak{P}(z)-\pi_0-\frac{\pi_1}{z}\right)+z\mathfrak{P}(z)\left(\mathfrak{G}(z)-b_0-\frac{b_1}{z}\right)\end{aligned}$$

Rearranging, we get

$$\begin{aligned}\mathfrak{P}(z) &= \frac{\pi_0(1-b_1)-b_0 z(\pi_0+\frac{\pi_1}{z})}{1-b_0 z-z\mathfrak{G}(z)+b_0 z} \\ &= \frac{\frac{\pi_0}{z}(1-b_1)-b_0(\pi_0+\frac{\pi_1}{z})}{\frac{1}{z}-b_0-\mathfrak{G}(z)+b_0}\end{aligned}$$

Notice that the boundary condition is at $z=1$, that is, $\mathfrak{P}(1)=1$. Therefore, applying L'Hôpital's rule, we get,

$$\lim_{z\to 1} P(z) = \lim_{z\to 1}\frac{\pi_0(1-b_1)-b_0\pi_1}{1-\mathfrak{G}'(z)} = \frac{\pi_0(1-b_1)-b_0\pi_1}{1-\mathfrak{G}'(1)}$$

where $\mathfrak{G}'$ is the derivative of $\mathfrak{G}$ with respect to $\frac{1}{z}$. Consequently,

$$\begin{aligned}\pi_0(1-b_1)-(1-a)\pi_0 &= 1-\mathfrak{G}'(1) \\ \therefore \pi_0 &= \frac{1-\mathfrak{G}'(1)}{a-b_1} \\ \therefore \pi_0 &= \frac{1-\mathfrak{G}'(1)}{b_0}\end{aligned}$$

where in the last step we used the fact that $a=b_0+b_1$. Next, we will evaluate $\mathfrak{G}'(1)$. We have

$$\begin{aligned}\mathfrak{G}(z) &= \sum_{j\geq 0} b_j z^{-j} \\ \mathfrak{G}'(z) &= \sum_{j\geq 1} j b_j z^{-(j-1)}\end{aligned}$$

where we differentiated $\mathfrak{G}(z)$ with respect to $\frac{1}{z}$. Thus, we have

$$\begin{aligned}
\mathfrak{G}'(1) &= \sum_{j\geq 1} j b_j \\
&= \sum_{j\geq 1} j \left[\mathrm{p}_Z^{j-1}\mathrm{p}_X^0 + \mathrm{p}_Z^j \mathrm{p}_X^1\right] \\
&= \sum_{j\geq 1} j \left[\frac{(\beta f)^{j-1} e^{-\beta f}}{(j-1)!} e^{-(1-\beta)f} + \frac{(\beta f)^j e^{-\beta f}}{j!}\left[1 - e^{-(1-\beta)f}\right]\right] \\
&= e^{-f} \sum_{j\geq 1} j \frac{(\beta f)^{j-1}}{(j-1)!} + \beta f e^{-\beta f}\left[1 - e^{-(1-\beta)f}\right] \sum_{j\geq 1} \frac{(\beta f)^{j-1}}{(j-1)!} \\
&\overset{(a)}{=} e^{-f}\left[e^{\beta f} + \beta f e^{\beta f}\right] + \beta f e^{-\beta f}\left[1 - e^{-(1-\beta)f}\right] e^{\beta f} \\
&= \beta f + e^{-(1-\beta)f}
\end{aligned}$$

where in (a), we used the fact that $xe^x + x = \sum_{j\geq 1} j \frac{x^{j-1}}{(j-1)!}$ and $e^x = \sum_{j\geq 0} \frac{x^j}{j!}$. Consequently,

$$\pi_0 = \frac{1 - \beta f - e^{-(1-\beta)f}}{e^{-\beta f}\left[1 - e^{-(1-\beta)f}\right]}$$

Now, for $f \to 0^+$, L'Hôpital's rule can be used to evaluate π_0, which gives $\pi_0 = \frac{1-2\beta}{1-\beta}$. Therefore, $\lim_{r\to\infty} P(S_i[r] = 0) = \frac{1-2\beta}{1-\beta}$ as $f \to 0^+$. This means that for sufficiently small f, the probability will be sufficiently close to $\frac{1-2\beta}{1-\beta}$. Furthermore, although the stationary distribution is defined as the round number $r \to \infty$, it will converge exponentially towards the stationary probability (see [147] for more details). This means that for a given f, at any round r after a warmup time of $T_{\text{warmup}} = \Theta(\lambda)$, the probability will be close to the stationary probability, and we can replace it with the stationary probability henceforth to simplify our computation.

C.4.3.2 Probability of $V_i^2[r]$

Next, we compute the probability of the event $V_i^2[r]$. Recall that $V_i^2[r]$ is the event as follows:

$$V_i^2[r] = \left\{ \exists t' > r \quad s.t. \quad \begin{matrix} Z_i[r,t']=0 \\ X_i[r,t'-1]=0 \\ Y_i[t']=1 \\ L_i[r+1,t] \geq 1 \quad \forall t > t' \end{matrix} \right\}$$

First, for a chain i and round $k > 0$, we define the following random walk:

$$D_i[k] = D_i[k-1] + X_i[k] - Z_i[k]$$
$$D_i[0] = 1$$

This random walk counts the difference between the number of honest blocks and number of adversarial blocks mined from round 1 to round k given that the random walk started from the initial state of $D_i[0] = 1$. Suppose that we start the random walk $\{D_i[k]\}_{k \geq 0}$ at round t'. Then,

- $Z_i[r, t'] = 0, X_i[r, t' - 1] = 0$ and $Y_i[t'] = 1$ implies that two honest blocks and no adversarial blocks were mined from the round r to round t'. Therefore, we set $D_i[0] = 1$.
- For any $k > 0$, $D_i[k] = L[r + 1, t' + k]$.

Now, $V_i^2[r] = \{D_i[0] = 1, D_i[k] \geq 1 \quad \forall k > 0\}$, and therefore, we get:

$$\begin{aligned} P(V_i^2[r]) &= \{D_i[0] = 1, D_i[k] \geq 1 \quad \forall k > 0\} \\ &= P(D_i[0] = 1) \times P\left(D_i[k] \geq 1 \quad \forall k > 0 \mid D_i[0] = 1\right) \end{aligned}$$

First, we compute $P(D_i[0] = 1)$ as

$$\begin{aligned} P(D_i[0] = 1) &= P\begin{pmatrix} \exists t' > r \text{ such that} \\ Z_i[r,t']=0 \\ X_i[r,t'-1]=0 \\ Y_i[t']=1 \end{pmatrix} \\ &= \sum_{m=1}^{\infty} P\begin{pmatrix} t'=r+m \\ Z_i[r,t']=0 \\ X_i[r,t'-1]=0 \\ Y_i[t']=1 \end{pmatrix} \\ &= \sum_{m=1}^{\infty} \left(e^{-(1-\beta)f}\right)^m \left(e^{-\beta f}\right)^{m+1} (1-\beta) f e^{-(1-\beta)f} \\ &= \frac{(1-\beta) f e^{-2f}}{1-e^{-f}} \end{aligned}$$

Now, we can use L'Hôpital's rule, to get $\lim_{f \to 0^+} P(D_i[0] = 1) = 1 - \beta$.

In order to compute $P\left(D_i[k] \geq 1 \quad \forall k > 0 \mid D_i[0] = 1\right)$, we first define

$$\mathbf{P}_u = P\left(D_i[r] \geq 1 \quad \forall r \geq 0 \mid D_i[0] = u\right)$$

Observe that the boundary conditions are $\mathbf{P}_0 = 0$ and $\mathbf{P}_\infty = 1$. Now for $u \geq 1$,

$$\begin{aligned} \mathbf{P}_n &= P\left(D_i[r] \geq 1 \quad \forall r \geq 0 \mid D_i[0] = u\right) \\ &= \sum_{k=-1}^{\infty} P\left(X_i[1] - Z_i[1] = -k, D_i[r] \geq 1 \quad \forall r \geq 0 \mid D_i[0] = u\right) \\ &= \sum_{k=-1}^{\infty} P\left(X_i[1] - Z_i[1] = -k\right) P\left(D_i[r] \geq 1 \quad \forall r \geq 0 \mid D_i[1] = u - k\right) \\ &= \sum_{k=-1}^{u-1} P\left(X_i[1] - Z_i[1] = -k\right) P\left(D_i[r] \geq 1 \quad \forall r \geq 0 \mid D_i[1] = u - k\right) \\ &= \sum_{k=-1}^{u-1} P\left(X_i[1] - Z_i[1] = -k\right) \mathbf{P}_{u-k} \\ &= a\mathbf{P}_{u+1} + c\mathbf{P}_u + \sum_{k=1}^{u-1} b_k \mathbf{P}_{u-k} \end{aligned}$$

where $a = P(X_i[1]-Z_i[1] = 1)$, $c = P(X_i[1]-Z_i[1] = 0)$ and $b_k = P(X_i[1]-Z_i[1] = -k)$. Therefore,

$$(1-c)\mathbf{P}_u z^{-u} = a\mathbf{P}_{u+1}z^{-u} + \sum_{k=1}^{u-1} b_k \mathbf{P}_{u-k} z^{-u}$$

$$(1-c)\sum_{u\geq 1}\mathbf{P}_u z^{-u} = az\sum_{u\geq 1}\mathbf{P}_{u+1}z^{-(u+1)} + \sum_{u\geq 1}\sum_{k=1}^{u-1} b_k \mathbf{P}_{u-k} z^{-u}$$

We now define the z-transforms $\mathfrak{P}(z) = \sum_{u\geq 1}\mathbf{P}_u z^{-u}$ and $\mathfrak{B}(z) = \sum_{k\geq 0} b_{k+1} z^{-k}$. We can skip $\mathbf{P}_0$ as $\mathbf{P}_0 = 0$. Therefore,

$$\begin{aligned}
(1-c)\mathfrak{P}(z) &= az(\mathfrak{P}(z) - \mathbf{P}_1 z^{-1}) + \sum_{k\geq 1}\sum_{u\geq k+1} b_k \mathbf{P}_{u-k} z^{-k} z^{-(u-k)} \\
&= a(z\mathfrak{P}(z) - \mathbf{P}_1) + \frac{1}{z}\sum_{k\geq 1} b_k z^{-(k-1)} \sum_{u\geq k+1} P_{u-k} z^{-(u-k)} \\
&= a(z\mathfrak{P}(z) - \mathbf{P}_1) + \frac{1}{z}\mathfrak{B}(z)\mathfrak{P}(z) \\
\therefore \mathfrak{P}(z) &= \frac{a\mathbf{P}_1 z}{az^2 - (1-c)z + \mathfrak{B}(z)}
\end{aligned}$$

Recall that,

- $a = P(X_i[1] - Z_i[1] = 1) = P(X_i[1] = 1)P(Z_i[1] = 0) = (1 - e^{-(1-\beta)f})e^{-\beta f}$
  $= e^{-\beta f} - e^{-f}$

- $c = P(X_i[1] - Z_i[1] = 0)$
  $= P(X_i[1] = 0)P(Z_i[1] = 0) + P(X_i[1] = 1)P(Z_i[1] = 1)$
  $= e^{-(1-\beta)f}e^{-\beta f} + (1 - e^{-(1-\beta)f})\beta f e^{-\beta f}$
  $= e^{-f} + \beta f(e^{-\beta f} - e^{-f})$

- $b_k = P(X_i[r] - Z_i[r] = -k)$
  $= P(X_i[r] = 0)P(Z_i[r] = k) + P(X_i[r] = 1)P(Z_i[r] = k+1)$

$$= e^{-(1-\beta)f}\frac{(\beta f)^k e^{-\beta f}}{k!} + (1 - e^{-(1-\beta)f})\frac{(\beta f)^{k+1}e^{-\beta f}}{(k+1)!}$$
$$= e^{-f}\frac{(\beta f)^k}{k!} + (e^{-\beta f} - e^{-f})\frac{(\beta f)^{k+1}}{(k+1)!}$$

Therefore, $\lim_{f\to 0^+} \mathfrak{B}(z) = 0$. Now, by L'Hôpital's rule,

$$\begin{aligned}\lim_{f\to 0^+} \mathfrak{P}(z) &= \frac{(1-\beta)\mathbf{P}_1 z}{(1-\beta)z^2 - z + \beta} \\ &= \frac{(1-\beta)\mathbf{P}_1}{\beta}\left[\left(\frac{1-\beta}{1-2\beta}\right)\left(\frac{1}{z^{-1} - \frac{1-\beta}{\beta}}\right) - \left(\frac{\beta}{1-2\beta}\right)\left(\frac{1}{z^{-1}-1}\right)\right] \\ &= \frac{(1-\beta)\mathbf{P}_1}{1-2\beta}\left[\left(\frac{1-\beta}{\beta}\right)\left(\frac{1}{z^{-1} - \frac{1-\beta}{\beta}}\right) - \left(\frac{1}{z^{-1}-1}\right)\right] \end{aligned} \tag{C.1}$$

To determine $\mathbf{P}_u$, we need to perform an inverse z-transform of $\mathfrak{P}(z)$ as shown below:

$$\mathbf{P}_u = \frac{1}{2\pi j}\oint_C \mathfrak{P}(z) z^{u-1} dz$$

where j is the imaginary unit, and C is a counterclockwise closed path encircling the origin and entirely lying in the region of convergence. Next, we want to determine $\lim_{f\to 0^+}\lim_{n\to\infty}\mathbf{P}_u$. However, we have,

$$\lim_{f\to 0^+}\lim_{u\to\infty}\mathbf{P}_u = \lim_{u\to\infty}\lim_{f\to 0^+}\mathbf{P}_u$$

wherein we used Moore-Osgood theorem [223] because:

- $\lim_{n\to\infty}\mathbf{P}_u$ exists pointwise for each different $f \neq 0$ because, by definition, $0 \leq \mathbf{P}_u \leq 1$ for all $u \geq 0$
- $\lim_{f\to 0^+}\mathbf{P}_u$ converges uniformly. This is because,
    1. The integral in the inverse z-transform is over a contour C that lies in the region of convergence and so $\mathfrak{P}(z)z^{u-1}$ is dominated by some Lebesgue integral function $\mathfrak{S}(z)$ in the sense that $|\,\mathfrak{P}(z)z^{u-1}\,| \leq \mathfrak{S}(z)$.

2. Therefore, by the dominated convergence theorem [223] and using equation C.1, we can interchange the limit and the contour integral to obtain

$$\lim_{f\to 0^+} \mathbf{P}_u = \lim_{f\to 0^+} \frac{1}{2\pi j} \oint_C \mathfrak{P}(z) z^{u-1} dz = \frac{1}{2\pi j} \oint_C \lim_{f\to 0^+} \mathfrak{P}(z) z^{u-1} dz$$
$$= \mathbf{P}_1 \left(\frac{1-\beta}{1-2\beta}\right) \left[1 - \left(\frac{\beta}{1-\beta}\right)^u\right].$$

Further, note that $0 \leq \mathbf{P}_1 \leq 1$.

3. Observe that $\mathbf{P}_1 \left(\frac{1-\beta}{1-2\beta}\right) \left[1 - \left(\frac{\beta}{1-\beta}\right)^u\right]$ uniformly converges to $\mathbf{P}_1 \left(\frac{1-\beta}{1-2\beta}\right)$.

Recall now, that our boundary condition was $P_\infty = 1$ i.e.

$$1 = \lim_{u\to\infty} \mathbf{P}_u = \lim_{f\to 0^+} \lim_{u\to\infty} \mathbf{P}_u = \mathbf{P}_1 \left(\frac{1-\beta}{1-2\beta}\right)$$

Finally, we get

$$\mathbf{P}_1 = \frac{1-2\beta}{1-\beta}$$

i.e., $\lim_{f\to 0^+} P\left(D_i[k] \geq 1 \quad \forall k > 0 \mid D_i[0] = 1\right) = \frac{1-2\beta}{1-\beta}$. Now, we can compute $P(V_i^2[r])$ as,

$$\lim_{f\to 0^+} P(V_i^2[r]) = \frac{1-2\beta}{1-\beta} \left(\lim_{f\to 0^+} P(D_i[0] = 1)\right)$$
$$= \frac{1-2\beta}{1-\beta}(1-\beta) = (1-2\beta)$$

C.4.3.3 Final computation of $\phi(f, \beta)$

Having computed the probabilities of $V_i^1[r]$ and $V_i^2[r]$, we can now complete the computation of $\phi(f, \beta)$ as,

$$\begin{aligned}\lim_{f \to 0^+} \phi(f, \beta) &= \lim_{f \to 0^+} P(V_i[r]) = \lim_{f \to 0^+} P(V_i^1[r]) \lim_{f \to 0^+} P(V_i^2[r]) \\ &= \frac{(1-2\beta)}{1-\beta}(1-2\beta) = \frac{(1-2\beta)^2}{1-\beta}\end{aligned}$$

This completes the proof of Lemma C.10. Now, since we want to bound β such that $\phi(f, \beta) > 0.5$ (for a sufficiently small f), we get $8\beta^2 - 7\beta + 1 > 0$. Recall that the Nakamoto consensus properties are not applicable for $\beta > \frac{1}{2}$. Therefore, we end up with the bound $\beta < \frac{7-\sqrt{17}}{16} \approx 0.1798$.

For our (τ, R)-respawning adversaries, recall that another τ fraction of nodes can get killed by the time a transaction is included in all parallel chains, and therefore, we will require $\beta + \tau < \frac{7-\sqrt{17}}{16}$. We also node that this bound may not be tight and that a larger adversarial threshold might still be acceptable.

C.4.4 Achieving cross-chain-quality

Equipped with the computation of $\phi(f, \beta) = P(V_i[r])$ from the previous section, we can proceed to calculate a bound on the number of chains d. For this, we first declare an event that needs to happen with negligible probability.

An unlikely event. Consider the following event T for any $\epsilon > 0$.

$$T := \left\{ \exists r \in \{1, \cdots, |\text{VIEW}|\} : \frac{1}{d} \sum_{i=1}^{d} \mathbf{1}_{V_i[r]} < \frac{1}{2} - \epsilon \right\}$$

The event T implies that if any transaction is revealed to the public in round r, then in at most $\frac{1}{2} - \epsilon$ fraction of the chains, an honest block will contain the transaction. In the following lemma, we show that if $\phi(f, \beta) > \frac{1}{2}$, then for sufficiently large d, $P(T)$ will be negligible in λ.

Lemma C.12. *Suppose β and a sufficiently small constant mining rate f are appropriately chosen such that $\phi(f, \beta) > \frac{1}{2}$. Then, $P(T) \leq |\mathsf{maxview}|\, e^{-\Omega(d)}$ where* maxview *is the longest view.*

Proof. Using the union bound,

$$\begin{aligned} P(T) &= P\left(\exists r \in \{1, \cdots, |\text{VIEW}|\} : \frac{1}{d} \sum_{i=1}^{d} \mathbf{1}_{V_i[r]} < \frac{1}{2} - \epsilon \right) \\ &\leq \sum_{1 \leq r \leq |\mathsf{maxview}|} P\left(\frac{1}{d} \sum_{i=1}^{d} \mathbf{1}_{V_i[r]} < \frac{1}{2} - \epsilon \right) \\ &= \sum_{1 \leq r \leq |\mathsf{maxview}|} P\left(\sum_{i=1}^{d} \mathbf{1}_{(V_i[r])^C} > \left(\frac{1}{2} + \epsilon \right) d \right) \end{aligned} \tag{C.2}$$

where $(V_i[r])^C$ is the complement of $V_i[r]$. Note that $P(\mathbf{1}_{(V_i[r]^C)}) = 1 - \phi(f, \beta)$. Since the mining for all chains is independent, we can consider $\mathbf{1}_{(V_1[r])^C}, \cdots, \mathbf{1}_{(V_g[r])^C}$ to be independent of each other. As $1 - \phi(f, \beta) < \frac{1}{2}$, therefore, for some $\delta > 0$, we can write

$$\begin{aligned} (1 + \delta) d (1 - \phi(f, \beta)) &= \left(\frac{1}{2} + \epsilon \right) d \\ \delta &= \frac{\left(\frac{1}{2} + \epsilon \right)}{1 - \phi(f, \beta)} - 1 \end{aligned}$$

Then, using the Chernoff bound on equation C.2, we have

$$\begin{aligned} P(T) &\leq \sum_{1 \leq r \leq |\mathsf{maxview}|} e^{-\frac{\delta^2 d[1-\phi(f,\beta)]}{3}} \\ &\leq \sum_{1 \leq r \leq |\mathsf{maxview}|} e^{-\Omega(d)} \\ &= |\mathsf{maxview}|\, e^{-\Omega(d)} \end{aligned}$$

□

Now, note that since $|\mathsf{maxview}|$ is polynomial in λ, setting $d = \Theta(\lambda)$ results in $P(T)$ being negligible in λ.

C.5 Soft-Ordering Application Proofs

Proof of Theorem 5.12. Suppose that a node N receives tx in round r. Then all honest nodes have received tx latest by round $r + \Delta$. Since node N received tx′ at round later than $r + 2\Delta$, that implies that tx′ was received at all other honest nodes earliest at round $r + \Delta + 1$. Therefore, $(\mathsf{tx}, \mathsf{tx}')$ is such that all honest nodes received tx before they received tx′, Since Π satisfies receive-order-fairness, tx will be ordered before tx′ in the final output log by all honest nodes. Consequently, Π will satisfy (2Δ)-soft-ordering. Note that the same proof also works for fair-linearizability (from Section 5.1.4).

Now, if N sees no conflicting transaction by round $r + 2\Delta$, then any conflicting transaction was received by an honest node at the earliest by round $r+\Delta+1$. Therefore, $(\mathsf{tx}, \mathsf{tx}')$ is such that all honest nodes received tx before they received tx′, Since Π satisfies receive-order-fairness, tx will be ordered before tx′ in the

final output log by all honest nodes. Consequently, Π will satisfy (2Δ)-soft-confirmation. □

Proof of Lemma 5.13. We can ignore the negligible number of bad views (for example, where soft-ordering is not satisfied). Suppose for contradiction that no Hamiltonian path exists that satisfies the property $\mathcal{Q}$. Now consider some Hamiltonian path P and its corresponding P^* (we know this exists since $\mathcal{S}_N^r$ is complete). If P^* is empty, then it is trivially a prefix of $\mathbb{L}_{\text{VIEW}}$. So consider a non-empty P^*.

Now, since P^* is not a prefix of $\mathbb{L}_{\text{VIEW}}$, then the following holds: There is a transaction tx ordered before tx′ in $\mathbb{L}_{\text{VIEW}}$ such that either (1) tx is ordered after tx′ in P^*; or (2) P^* contains tx′ but not tx. We will proceed by a simple recursive argument on the number of such transaction pairs.

Suppose that there is only one pair (tx, tx') that breaks the prefix property. If (1) is true, then either $\mathcal{S}_N^r$ contains a unidirectional edge (tx', tx) or it contains edges in both directions. However, since Π satisfies δ-soft-ordering, the unidirectional edge would imply the ledger always contains tx′ earlier. Consequently, $\mathcal{S}_N^r$ needs to contain a bidirectional edge between the two transactions. This means that there is another path that simply flips the position of tx and tx′, and this would now be a prefix of $\mathbb{L}_{\text{VIEW}}$.

If (2) is true, then this means tx′ was received more than δ rounds before tx (since tx′ was present in P^* but tx wasn't). But this implies that $\mathbb{L}_{\text{VIEW}}$ must contain tx′ first which is a contradiction.

Now, given any path P_0, and the corresponding P_0^* that conflicts with $\mathbb{L}_{\text{VIEW}}$ in k transaction pairs, we can use the above technique to arrive at another

Hamiltonian path P that satisfies the property $\mathcal{Q}$. At every step of the recursion, the number of conflicting transaction pairs will be reduced by at least 1, i.e., the recursion will take at most k steps

At the end of the recursion, we get a path P which satisfies $\mathcal{Q}$, and therefore contradicts our assumption. This completes this proof. □

APPENDIX D

APPENDIX FOR CHAPTER 6

D.1 Security Proofs for Themis

In this section, we formally prove the security properties of Themis.

D.1.1 Helper Lemmas

We begin with some helpful lemmas.

Lemma D.1 (Dependency Graph Properties)**.** *The following hold for the output of the algorithm* `FairPropose(`$\mathcal{L}$`)`*: (1) It contains all solid transactions; (2) For all solid* tx *and non-blank* tx′ *contained in the output, exactly one of the edges* $(\mathsf{tx}, \mathsf{tx}')$ *and* $(\mathsf{tx}', \mathsf{tx})$ *is present in the output graph.*

Proof. Let $\mathcal{G}$ be the dependency graph constructed by the algorithm `FairPropose(`$\mathcal{L}$`)`. It is easy to see why (1) holds since the final graph output only removes those SCCs from $\mathcal{G}$ that do not contain a single solid transaction.

To prove (2), first notice that since tx is solid and tx′ is non-blank, at least $n - 2f > 2n(1-\gamma) + 2f$ orderings in $\mathcal{L}$ contain at least one of tx or tx′, i.e., order the two transactions. Therefore, at least one of $\mathsf{tx} \prec_{\mathcal{L}, n(1-\gamma)+f+1} \mathsf{tx}'$ or $\mathsf{tx} \prec_{\mathcal{L}, n(1-\gamma)+f+1} \mathsf{tx}'$ holds. Since only one edge is added even when both holds, we conclude that exactly one of the edges $(\mathsf{tx}, \mathsf{tx}')$ and $(\mathsf{tx}', \mathsf{tx})$ is contained in the final graph output. □

Lemma D.2. *If* `FairPropose(`$\mathcal{L}$`)` *outputs a solid transaction* tx *and does not output* tx′ *(and* tx′ *has not been previously proposed), then there are at least* $n(1-\gamma)+1$ *honest nodes that have received* tx *before* tx′*.*

Proof. If tx′ is blank, since tx′ is present in at most $n(1-\gamma)+f$ orderings, we have tx $\prec_k$ tx′ where $k = n - 2f - n(1-\gamma) - f = \gamma n - 3f > n(1-\gamma)+f$ since $n > \frac{4f}{2\gamma-1}$. Equivalently, since at most f of those are adversarial, it holds that more than $n(1-\gamma)+1$ honest nodes have received tx before tx′. □

Lemma D.3 (Contiguous Cycles)**.** *If* `FairPropose(`$\mathcal{L}$`)` *outputs* tx *and does not output* tx′ *(and* tx′ *has not been previously proposed), then either* tx *is an earlier cycle than* tx′ *or* tx *and* tx′ *are in the same cycle. Further, if* tx′ *is received before* tx *by* γn *nodes, then* tx *and* tx′ *are in the same cycle.*

Proof. We first show that if tx is output and tx′ is not, then it cannot be the case that tx′ is in an earlier cycle. To see why, assume for contradiction that tx′ is an earlier cycle. This means that there is no sequence of transactions $[\mathsf{tx} = \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_l = \mathsf{tx}']$ such that for each j it holds that tx_j was received before tx_{j+1} by $n(1-\gamma)+1$ honest nodes. But this contradicts the fact that tx was output and tx′ was not, and therefore we conclude that tx′ cannot be in an earlier cycle.

Now, for the case when tx′ is received before tx by γn nodes, note that tx cannot be solid within the leader proposal since this would imply that there are at least $n(1-\gamma)+1$ honest nodes that have received tx before tx′ (from Lemma D.2), which contradicts the given condition. Therefore tx must be shaded and consequently, it was included in the proposal since there is a sequence of transactions $[\mathsf{tx} = \mathsf{tx}_0, \mathsf{tx}_1, \ldots, \mathsf{tx}_l]$ such that the graph contains each of the edges $(\mathsf{tx}_i, \mathsf{tx}_{i+1})$ and tx_l is a solid transaction. This also means that tx_l was received before tx′ by

at least $n(1-\gamma)+1$ honest nodes.

Now, since tx′ is received before tx by γn nodes, this directly implies that $[\text{tx}_0, \text{tx}_1, \ldots, \text{tx}_l, \text{tx}']$ is a cycle. □

We are now ready to prove the main theorems for Themis.

D.1.2 Proof of Themis Fairness (Theorem 6.4)

To show batch-order-fairness, consider tx and tx′ such that tx was received before tx′ by at least γn nodes. Now, we follow a few cases:

(Case 1) At some time, the current correct leader proposal includes tx and tx′ is not included in this or any earlier proposal. Now, by Lemma D.3, we know that tx′ must be in the same or later cycle as tx.

(Case 2) At some time, the current correct leader proposal includes tx′ and tx is not included in this or any earlier proposal. Since we are also given that tx was received before tx′ by γn nodes, by Lemma D.3, we know that tx′ must be in the same cycle as tx.

(Case 3) At some time the current correct leader proposal includes both tx and tx′. Since tx was received before tx′ by γn nodes, we know that there is an edge from tx to tx′ within the dependency graph. Therefore, once again, either tx′ will be output in the same cycle as tx or a later one.

The above cases show that the final transaction ordering final transaction ordering can be split into contiguous cycles. In other words, either tx will be output before tx′ or some contiguous transaction sequence containing both tx and tx′

will be a cycle. Consequently, there are indices $1 = i_1 < \cdots < i_k = l + 1$ such that $[C_1, \ldots, C_{k-1}]$, with batches $C_j = \{\mathsf{tx}_{i_j}, \ldots, \mathsf{tx}_{i_{j+1}-1}\}$, satisfies γ-batch-order-fairness.

Now, to show consequent-transaction fairness, in the output ordering, consider transactions tx_j and tx_{j+1} where tx_{j+1} is output immediately after tx_j. The following cases arise:

(Case 1) Both transactions were part of the same initial leader proposal. Now, if they were part of the same SCC, then since the output order is a Hamiltonian cycle, $(\mathsf{tx}_j, \mathsf{tx}_{j+1})$ is an edge in the graph; in other words, tx_j was received before tx_{j+1} by $n(1-\gamma)+1$ honest nodes. On the other hand, if tx_j was in an earlier SCC, then $(\mathsf{tx}_j, \mathsf{tx}_{j+1})$ is already an edge. Since tx_j was output earlier, it cannot be in a later SCC.

(Case 2) tx_{j+1} was proposed in the block after tx_j was proposed. This means that, in some valid leader proposal, tx_j was present but tx_{j+1} was not, and further tx_j was the last transaction in the previous proposal, which implies that tx_j was a solid transaction. Therefore, we can directly conclude the result by Lemma D.2.

(Case 3) Note that it not possible for tx_j to be proposed in a block after tx_{j+1} since the final ordering contains tx_j first.

It is easy to see that, similar to Aequitas, the batches are minimal in the sense that they are of size one when there are no cycles. Consequently, in such a case, Themis satisfies the stronger receive-order-fairness notion.

D.1.3 Proof of Themis Consistency and Liveness (Theorem 6.5)

We now show that Themis satisfies the satandard SMR properties of consistency and (standard) liveness.

Consistency is a direct consequence of the consistency of the underlying consensus algorithm. For liveness, consider a transaction tx that has been received by all nodes, i.e., it will be present in at least $n - 2f$ local replica orderings sent to the leader. In other words, tx is solid, and will be included in the leader's proposal. Note that the final ordering of tx only depends on earlier (shaded) transactions that have been proposed by the current or an earlier proposal (since this can cause edges to be missing in the graph proposal). Note that the final ordering of tx no longer depends on any transactions that are not yet proposed. This means that as soon as the edges between the previously shaded transactions are added (this will happen when the shaded transactions are received by enough nodes which is only dependent on the network delay). Consequently, Themis achieves standard liveness.

D.2 Themis Experiment Details

Toy AWS setting. We start with a simple geo-distributed network with one node each in five different AWS regions: us-west-1 (California), us-east-2 (Ohio), ap-northeast-1 (Tokyo), ap-northeast-2 (Seoul), and eu-central-1 (Frankfurt) and measured ping timings between each pair of servers (in both directions) as a proxy for the communication latency between them. Table D.1 contains ping times averaged over 1000 samples.

Origin \ End	us-west-1 (California)	us-east-2 (Ohio)	ap-northeast-1 (Tokyo)	ap-northeast-2 (Seoul)	eu-central-1 (Frankfurt)
us-west-1	-	51.407	105.733	133.771	147.308
us-east-2	51.166	-	131.609	159.931	98.827
ap-northeast-1	106.684	131.359	-	31.723	228.356
ap-northeast-2	134.018	159.561	32.397	-	222.829
eu-central-1	145.962	97.972	228.862	234.641	-

Table D.1: Average ping times (in ms) between different AWS servers

Out of 20 total node pairs (A, B), we found that 10 are frontrunnable (i.e., A can frontrun transactions originating from B) in the case of fair separability. As a concrete example based on Table D.1, an adversary in Tokyo could frontrun a user transaction from Seoul by sending its adversarial transaction to California before the transaction from Seoul reached Frankfurt. On the other hand, there were no instances of triangle inequality being broken in our toy setting, and as a consequence, there were no frontrunnable pairs in the case of order-fairness.

Proof of Lemma 7.3. To see why, consider a transaction dependency graph $\mathcal{G}^*$ that contains an edge from tx to tx′ is a *majority* of nodes claim to have received tx earlier. First note that this graph is complete (i.e., there is one edge between any transaction pair), and that any cycle in the actual dependency graph will also be a cycle in this graph.

Now, since $\mathcal{G}^*$ is complete, if a cycle within $\mathcal{G}^*$ contains m, then m is also contained within a cycle of length 3, i.e., there exists some x, y such that $x \to y \to m \to x$ is a cycle. Note that this is directly true for Themis since its dependency graph is already complete.

Now, if x, y are both honest user transactions, then by our network assumption, all honest users have received y before m and m before x. This implies that no honest node will have received x before y and so the cycle cannot exist. On

the other hand, w.l.g, if only x is honest, then again by our network assumption, all honest users must have received m before x and x before y. Therefore, no honest node has received y before m and therefore again the cycle cannot exist. □

D.3 Additional Insights on Related Works

Comparing the graph structure of Aequitas and Themis. For some inputs, both Aequitas and Themis build the dependency graph in the same way. In other cases, Themis seems to approximate the graph structure for Aequitas with a smaller γ (since Themis will always attempt to add an edge between any transaction pair to construct, while some edges may be missing from the Aequitas graph). However, to extract a total ordering, Aequitas still needs to compare transactions without an edge, for which it uses the number of descendants in the graph. In Themis, the comparison between such transactions is done explicitly by adding the edge corresponding to the larger weight. This reveals that although both protocols satisfy batch-order-fairness, there are likely other subtleties within the graph structure that are not captured by the definition. We leave the problem of uncovering these to future work.

Equivalence of differential-order-fairness [54] and batch-order-fairness. The following theorem shows an equivalance between the two definitions:

Theorem D.4. *Consider specific n, f. A protocol satisfies κ-differential-order-fairness ($\kappa \geq 0$) if and only if it also satisfies γ-batch-order-fairness where $\gamma \geq \frac{1}{2} + \frac{3f+1+\kappa}{2n}$.*

Proof. Define $b(\mathsf{tx}, \mathsf{tx}')$ to be the number of honest nodes that receive tx before

tx′. Notice that γ-batch-order-fairness implies γ'-batch-order-fairness for any $\gamma \leq \gamma' \leq 1$.

($\Longrightarrow$). Suppose that a protocol Π satisfies γ^*-batch-order-fairness for $\gamma^* \in (\frac{1}{2}, 1]$ such that Π does not satisfy the property for any $\gamma < \gamma^*$. Now, consider any transactions $\mathsf{tx}_1, \mathsf{tx}_2$ such that $b(\mathsf{tx}_1, \mathsf{tx}_2) - b(\mathsf{tx}_2, \mathsf{tx}_1) > \kappa + 2f$. Since we have $b(\mathsf{tx}_1, \mathsf{tx}_2) + b(\mathsf{tx}_2, \mathsf{tx}_1) \geq n - f$, adding the two, we get $2 \cdot b(\mathsf{tx}_1, \mathsf{tx}_2) \geq n + f + \kappa + 1$, or equivalently, $b(\mathsf{tx}_1, \mathsf{tx}_2) \geq \frac{n+f+\kappa+1}{2}$.

Now, since Π satisfies γ^*-batch-order-fairness where γ^* is minimal, if $\frac{n+f+\kappa+1}{2} = \gamma^* n - f$, then Π will order tx_1 no later than tx_2, i.e., it will also satisfy κ-differential-order-fairness. This implies $\gamma^* = \frac{1}{2} + \frac{3f+1+\kappa}{2n}$.

($\Longleftarrow$). Suppose that a protocol Π satisfies κ-differential-order-fairness. Define $\gamma^* = \frac{1}{2} + \frac{3f+1+\kappa}{2n}$. Now, consider two transactions tx_1 and tx_2 such that $\gamma^* n$ nodes have received tx_1 before tx_2. This means that $\gamma^* n - f = \frac{n+f+1+\kappa}{2}$ honest nodes have received tx_1 before tx_2, i.e., $b(\mathsf{tx}_1, \mathsf{tx}_2) \geq \frac{n+f+1+\kappa}{2}$. Therefore, $b(\mathsf{tx}_2, \mathsf{tx}_1) \leq (n-f) - \frac{n+f+1+\kappa}{2} = \frac{n-3f-\kappa-1}{2}$. Consequently, $b(\mathsf{tx}_1, \mathsf{tx}_2) - b(\mathsf{tx}_2, \mathsf{tx}_1) \geq 2f + \kappa + 1$, i.e., $b(\mathsf{tx}_1, \mathsf{tx}_2) > b(\mathsf{tx}_2, \mathsf{tx}_1) + 2f + \kappa$ which means that Π will deliver tx_1 no later than tx_2. Therefore, Π satisfies γ^*-batch-order-fairness, and in turn also for any $\gamma \geq \gamma^*$. □

Remark 22 (κ parameterization)**.** First notice that the $\kappa = 0$ and $n = 3f + 1$ case corresponds to the $\gamma = 1$ case for batch-order-fairness. Note that the protocol from [54] works for $n = 3f + 1$ only when all nodes are honest, and in this setting, Aequitas and Themis both also satisfy $\gamma = 1$ batch-order-fairness.

For a given κ, if tx_1 and tx_2 are such that $b(\mathsf{tx}_1, \mathsf{tx}_2)$ and $b(\mathsf{tx}_2, \mathsf{tx}_1)$ differ by more than $\kappa + 2f$, then their fairness is "considered" by the protocol. To provide

the same fairness as batch-order-fairness for a $\gamma = \frac{1}{2} + \epsilon$ where ϵ is small, the minimum acceptable difference between $b(\mathsf{tx}_1, \mathsf{tx}_2)$ and $b(\mathsf{tx}_2, \mathsf{tx}_1)$ needs to also be made small, for which the only option is to make f much smaller than n. This corresponds exactly to the bound for batch-order-fairness where the corruption threshold reduces as γ tends to $\frac{1}{2}$.

As an example, when $n = 3f + 1$, the only κ that makes sense is $\kappa = 0$, which corresponds only to $\gamma = 1$ (i.e., the weakest version) in the batch-fairness context. This means that to get a stronger fairness property, the only option is to reduce the corruption ratio f/n and this will in turn also correspond to a stronger γ (i.e., closer to 1/2).

APPENDIX E

APPENDIX FOR CHAPTER 9

E.1 $\mathcal{G}_{\mathsf{SGX}}$ Functionality

We detail the full $\mathcal{G}_{\mathsf{SGX}}$ functionality in Fig. E.1.

$\mathcal{G}_{\mathsf{SGX}}[S, \mathsf{Reg}]$

On initialization: $(\mathsf{mpk}, \mathsf{msk}) \leftarrow\!\!\$\ \mathsf{S.kg}(1^\lambda), I \leftarrow \emptyset$.

On receive $\texttt{getpk}^*()$ from some party $\mathcal{M}$:
- Send mpk to $\mathcal{M}$.

On receive $\texttt{install}^*(idx, \mathsf{prog})$ from some $\mathcal{M} \in \mathsf{Reg}$:
- If $\mathcal{M}$ is honest, assert $idx = sid$.
- Generate nonce $eid \in \{0,1\}^\lambda$.
- Store $I[eid, \mathcal{M}] = (idx, \mathsf{prog}, 0)$ and send eid to $\mathcal{P}$.

On receive $\texttt{resume}^*(eid, \mathsf{inp})$ from some $\mathcal{M} \in \mathsf{Reg}$:
- Let $(idx, \mathsf{prog}, \mathsf{mem}) = I[eid, \mathcal{M}]$, abort if not found.
- Compute $(\mathsf{out}, \mathsf{mem}') = \mathsf{prog}(\mathsf{inp}, \mathsf{mem})$.
- Update $I[eid, \mathcal{M}] = (idx, \mathsf{prog}, \mathsf{mem}')$.
- Let $\sigma = \mathsf{S.sign}_{\mathsf{msk}}((idx, eid), \mathsf{prog}, \mathsf{out})$.
- Send (out, σ) to $\mathcal{M}$.

Figure E.1: $\mathcal{G}_{\mathsf{SGX}}$ global functionality from [209]. Starred operations are re-entrant activation points.

E.1.1 Security Proof for SGX-PoCK

We now provide the full details of the CK-soundness proof for SGX-PoCK.

CK-Soundness proof. Let game G_1 denote the CK-soundness game, where $\mathcal{A}$ must convince $\mathcal{V}$ without the witness being extracted by $\mathcal{E}$. In particular,

given $x \in L$ (with witness w) and pp, for adversary $\mathcal{A}$ using resource $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$, we define the CK-soundness advantage, i.e., the probability of winning G_1 as follows:

$$\mathsf{CK\text{-}Adv}(\mathcal{A}^{\mathcal{R}'}) = \Pr\left[\begin{smallmatrix}\langle \mathcal{A}^{\mathcal{R}'}(pp,x), \mathcal{V}(pp,x)\rangle = 1 \\ \wedge\ (x, w' = \mathtt{out}_{\mathcal{E}}) \notin R_L\end{smallmatrix}\right].$$

Observe first that the CK-soundness definition holds whenever CK-Adv is bounded by $\mathrm{negl}(\lambda)$. We will show that for any $\mathcal{A}$ using a "practically instantiable" resource $\mathcal{R}' \in \mathfrak{R}_{\mathsf{all}}$, its CK-Adv is negligibly small. First, observe that $\mathcal{V}$ outputs 1 only on a valid signature from the honest resource $\mathcal{R} = \mathcal{G}_{\mathsf{SGX}}$, i.e., to convince $\mathcal{V}$, it must be the case that $\mathcal{A}^{\mathcal{R}'}$ constructed a valid $\mathcal{G}_{\mathsf{SGX}}$-signature on w. We will use this to bound CK-Adv by the SUF-CMA security of the signature scheme S—here, $\mathcal{G}_{\mathsf{SGX}}$ is instantiated with the key sampled by the challenger in the SUF-CMA game.

Suppose that $\mathcal{A}$'s usage of $\mathcal{R}'$ results in q queries made to $\mathcal{G}_{\mathsf{SGX}}$. Without loss of generality, we can assume that any queries to $\mathcal{G}_{\mathsf{SGX}}$ are not made directly by $\mathcal{R}'$ but instead made through $\mathcal{A}$ as instructed by $\mathcal{R}'$ (i.e., $\mathcal{A}$ has access to two resources—$\mathcal{R}'$ and $\mathcal{G}_{\mathsf{SGX}}$).

Now let game G_2 be the same as G_1 except now, if (x, w) ($\in R_L$) is queried to $\mathcal{G}_{\mathsf{SGX}}$, $\perp$ is output instead of the signature; denote this by $\mathcal{G}^*_{\mathsf{SGX}}$. Notice that

$$\Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_1] = \Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_2]$$

since whenever $\mathcal{A}$ queries $\mathcal{G}_{\mathsf{SGX}}$ with w, it will always be output by $\mathcal{E}$.

Let game G_3 be the same as G_2 except that the part with the extractor is removed (i.e., $\mathcal{A}$ wins if it just convinces $\mathcal{V}$). It is easy to see that

$$\Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_2] \leq \Pr[\mathcal{A}^{\mathcal{R}', \mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_3]$$

since the adversary wins in G_3 whenever it wins in G_2.

Now, with the assumption that no practical resource in $\mathfrak{R}_{\mathsf{all}}$ (intuitively other than $\mathcal{G}_{\mathsf{SGX}}$) can give additional information about the signatures (this is equivalent to assuming that $\mathcal{R}'$ can be simulated without access to the msk of $\mathcal{G}_{\mathsf{SGX}}$), we can remove any additional power given by $\mathcal{R}'$. In particular, consider game G_4 where any queries to $\mathcal{R}'$ return $\perp$; denote this resource by $\mathcal{R}'_{\perp}$. Under the aforementioned assumption, it holds that

$$\Pr[\mathcal{A}^{\mathcal{R}',\mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_3] = \Pr[\mathcal{A}^{\mathcal{R}'_{\perp},\mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_4].$$

Further, since $\mathcal{R}'_{\perp}$ only returns $\perp$, we can construct $\mathcal{B}$ which never queries $\mathcal{R}'$ such that

$$\Pr[\mathcal{A}^{\mathcal{R}'_{\perp},\mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_4] \leq \Pr[\mathcal{B}^{\mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_4].$$

Recall from earlier that this $\mathcal{G}^*_{\mathsf{SGX}}$ returns $\perp$ when queried with the witness w and that $\mathcal{B}$ wins in G_4 if it convinces $\mathcal{V}$, i.e., outputs a valid signature on w. Now, let game G_5 be the standard SUF-CMA security game for the signature scheme S. We can construct an adversary $\mathcal{C}$ for this game as follows: $\mathcal{C}$ runs $\mathcal{B} \leftrightarrow \mathcal{V}$ internally and forwards any queries to $\mathcal{G}^*_{\mathsf{SGX}}$ from $\mathcal{B}$ to its own SUF-CMA signing oracle. If $\mathcal{V}$ outputs 1, then $\mathcal{C}$ outputs the last message sent by $\mathcal{B}$ as its own SUF-CMA forgery. Notice now that $\mathcal{C}$ wins the SUF-CMA game exactly when $\mathcal{B}$ convinces $\mathcal{V}$—i.e., it holds that

$$\Pr[\mathcal{B}^{\mathcal{G}^*_{\mathsf{SGX}}} \text{ wins in } \mathsf{G}_4] = \Pr[\mathcal{C} \text{ wins in } \mathsf{G}_5] = \mathsf{SUF\text{-}CMA}_S(\mathcal{C}).$$

Further, $\mathcal{C}$ makes exactly as many queries to its signing oracle as $\mathcal{B}$ makes to $\mathcal{G}_{\mathsf{SGX}}$.

Finally, since the signature scheme S being used is SUF-CMA secure, we know that the adversarial advantage for any adversary is bounded by some

$\mathsf{negl}(\lambda)$ which in turn implies that the $\mathsf{CK\text{-}Adv}$ for any $\mathcal{A}$ and $\mathcal{R}'$ is bounded by $\mathsf{negl}(\lambda)$. This completes the proof.

Making $\mathsf{SGX\text{-}PoCK}$ satisfy zero-knowledge. Intuitively, to make the protocol zero-knowledge, there must exist a simulator $\mathcal{S}$ that can simulate the protocol transcript without access to the witness. Further note that $\mathcal{S}$ should also not be able to program the master secret key of the SGX (this is accounted for since we model the SGX resource as a GUC functionality).

We now briefly describe how $\mathsf{SGX\text{-}PoCK}$ can be made zero-knowledge. As is the case with other GUC protocols, $\mathcal{S}$ will require a trapdoor in other to simulate transcripts. Towards this, intuitively, we introduce a trapdoor τ which allows the generation of arbitrary SGX attestations for our protocol.

Specifically, $\mathcal{V}$ first chooses a trapdoor τ and submits it to $\mathcal{G}_{\mathsf{SGX}}$ which returns an attestation σ_c on $c = \mathsf{owf}(\tau)$ where owf is a one-way function. $\mathcal{V}$ sends the (c, σ_c) to $\mathcal{P}$ who then submits (c, x, w) to $\mathcal{G}_{\mathsf{SGX}}$. Before providing an attestation that the witness w was seen, $\mathcal{G}_{\mathsf{SGX}}$ ensures that the correct c was input. Finally, the $\mathcal{V}$ can check the correctness of the attestation to complete the proof. Note that this protocol also requires $\mathcal{V}$ to possess an SGX device.

This construction will now be zero-knowledge since the simulator $\mathcal{S}$ can use the trapdoor τ to forge attestations and simulate the interaction between $\mathcal{P}$ and $\mathcal{V}$. The proof is analagous to the one from [209, Thm. 19].

Remark 23 (SGX inside SGX)**.** $\mathsf{SGX\text{-}PoCK}$ can in fact be modified to work even when an SGX can be run inside another SGX as long as this can be done only a finite number of times. Specifically, if is practical to run k layers of SGX but not $k + 1$ layers, then $\mathsf{SGX\text{-}PoCK}$ can be modified to use the k-layer SGX as the

resource $\mathcal{R}$; $\mathcal{P}$ now obtains an attestation from this $\mathcal{R}$. By assumption, since a $(k+1)$-SGX is not practical, the witness submitted to $\mathcal{R}$ will be seen by $\mathcal{E}$ allowing for extraction.

This also serves to future proof our protocol in case of advances in TEE infrastructure.

E.2 Deferred Details for ASIC-ZKPoCK

E.2.1 Proofs

ASIC-ZKPoCK CK-soundness proof. Similar to the SGX-PoCK CK-soundness proof, let G_1 denote the CK-soundness game where $\mathcal{A}$ must convince $\mathcal{V}$ (in more than y out of n rounds) without the witness being extracted by $\mathcal{E}$ and define $\mathsf{CK\text{-}Adv}(\mathcal{A}^{\mathcal{R}'})$ similarly as before given $\mathcal{A}$ and resource $\mathcal{R}'$.

Now define game G_2 to be exactly as G_1 except that only the first query made (by e.g., $\mathcal{A}$ or $\mathcal{R}'$) to each honest resource in $\mathfrak{R}$ is returned correctly; all other queries return $\perp$. Further, define game G_3 similarly except that all queries to resources in $\mathfrak{R}$ are returned as $\perp$. Recall that making two queries to an ASIC resource allows $\mathcal{E}$ to extract the witness because of the quasi-soundness of the sigma protocol. Therefore, we have:

$$\Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_1] \leq \Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_2] + \Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_3]$$

Now, for G_2, suppose that the adversary deploys one challenge each to m different ASICs on different machines to not have two queries to the same ASIC.

While we do analyze this strategy in the security proof, we note that this actually requires the adversary to enforce strong non-collusion requirements in order to succeed, which is likely infeasible; see Remark 14 for details. For small enough $m = \Theta(1)$, we can set parameters such that the probability that $\mathcal{A}$ wins in G_2 is negligible in λ. To see why, note that similar to Lemma 9.3, the probability that m challenges are sufficient (in 1 round) is $p = 1 - (1 - 1/d)^{\beta m}$. For small enough m, we can set $\beta = \Theta(1)$ such that the $p < y/n$, and therefore $\Pr[\mathcal{A}^{\mathcal{R}'} \text{ wins in } \mathsf{G}_2]$ is negligible in λ.

Now, the probability that $\mathcal{A}$ wins in G_3 is exactly the probability that it wins without using the ASIC resource. Assume that there is no practical resource that is encrypted (or in other ways prevents eavesdropping by $\mathcal{E}$; this also prevents for instance, fast ASICs that can compute hashes in MPC with secret-shared inputs) and computes hashes faster than $Q_{\text{adv}} = mQ_{\mathsf{cpu}}$ (i.e., equivalent to a network of m CPUs). We note that although their setting is simpler, [107] also makes a qualitatively identical assumption by modeling all MPC-based hash queries as "slow" as opposed to "fast" ASIC queries. Consequently, the probability that $\mathcal{A}$ wins in $\mathcal{G}_2$ can be bounded using Lemma 9.2 as follows: Suppose that parameters are set such that $p_{\mathsf{one}}^{Q_{\text{adv}}} < y/n = \Theta(1)$. From Fact 9.1, it is easy to see that the the probability $\mathcal{A}$ succeeds in more than y rounds, i.e., wins in game G_3 is negligible in λ.

Finally, this means that we can bound the CK-Adv by $\text{negl}(\lambda)$ which completes the proof.

ASIC-ZKPoCK **HVZK proof.** We now provide additional details on the honest-verifier zero-knowledge (HVZK) proof.

Recall that the proof $\pi = (B, \nu)$ sent to $\mathcal{V}$ contains only a single Σ-protocol transcript. We can define the zero-knowledge simulator $\mathcal{S}$ as follows: Given the public parameters pp, and the statement x, first randomly sample r. Intuitively, $\mathcal{S}$ will now use the ASIC-resource $\mathcal{R}$ to randomly sample $B[1]$ until a satisfying PoW solution ν is found. In essence, $\mathcal{S}$ will act the same way as the honest prover $\mathcal{P}$ except in the computation of the Σ-protocol transcript (a, c, s)—instead of using the pre-committed a and the challenge c to compute s, it will run the simulator from the Σ-protocol to generate (a, s) given c.

Formally, given $B[1]$, let $c = \mathcal{H}(B[1] \parallel r)$. Run the simulator for the Σ-protocol on c to obtain a simulated transcript (a, c, s) for knowledge of a witness for x; this is possible since the Σ-protocol is also HVZK. Finally, use $\mathcal{R}$ to find whether there exists a nonce ν for the PoW puzzle, i.e., satisfying $\mathcal{H}(B[1] \parallel B[2] = s, \nu) < 2^{\ell}/d$. Note that ℓ and d are given in the parameters pp. If such a ν does not exist, then a different $B[1]$ is sampled and the process is repeated.

E.2.2 Parameterization Details

Estimating Q_{asic} and Q_{cpu}. A top-of-the-line mining ASIC for Bitcoin, the Antminer S19 Pro Hydro, released in 2022, has a rated performance of 154 TH/s [33], i.e., about $Q_{\mathsf{asic}} \approx 2^{47}$ H/s. Each hash in this case is a "Bitcoin" hash: a double invocation of SHA-256 on two 64-byte input blocks, as required for Bitcoin mining. The set $\mathfrak{R}$ of satisfying resources for our formalism will therefore consist of all ASICs with a hash rate of at least Q_{asic}.

Even under optimistic assumptions (including use of native hardware sup-

port for SHA), an SGX application[1] on a state-of-the-art 4.60 GHz Intel processor can execute at most about 72 MH/s, i.e, $Q_{\mathsf{cpu}} \approx 2^{26}$ H/s (where hashes here are "Bitcoin" hashes) [212].

Example practical parameterization. Given $Q_{\mathsf{asic}} \approx 2^{47}$ H/s and $Q_{\mathsf{cpu}} \approx 2^{26}$ H/s, we might for instance target a forced revelation error and completeness error of $\epsilon < 10^{-6}$. We set $y = \lfloor n/2 \rfloor$ for the most efficient parameterization. While, the error bounds are also achieved for much smaller value of τ, we set the duration of each round τ to 5 seconds, allowing for any network communication delays. Difficulty $d = 2^{47}$ and number of rounds $n = 8$ allow the ASIC to succeed with probability more than $1 - \epsilon$ while an adversary using a network of 10,000 CPUs only succeeds with probability less than ϵ. Note that, the range of nonce (security parameter β) can be set appropriately to minimize p^{onechal}. For example, by Lemma 9.3, a nonce of length 5 bytes ($\beta = 2^{40}$) would let an adversary which uses only one challenge, succeed with probability $<< \epsilon$. Figure 9.5 shows the completeness probability of the ASIC and the success probability of an adversary with 10,000 CPUs as a function of the difficulty and number of rounds (each round is set to 5 secs).

With a total execution time of 50s, it is possible to achieve an overwhelming probability of completeness as well as an overwhelming probability of failure for an adversary trying to bypass forced revelation with a network of 10,000 CPUs.

Concrete ASIC-ZKPoCK implementation details. We implement the ASIC-ZKPoCK prover using an outmoded Antminer S9 ASIC and the verifier using

[1]Note that secure use of SGX requires disablement of hyperthreading.

an Ethereum Smart Contract. While this ASIC has a smaller hashrate compared to the latest hardware in the market, and the Ethereum network inherently has a coarse granularity for measuring time, we can still achieve reasonable error probabilities for completeness and forced revelation. Below is one such example parameterization for our implementation:

- $Q_{\mathsf{asic}} = 13\text{TH/sec} \sim 2^{43.5}$.
- $Q_{\mathsf{cpu}} = 2^{26}$.
- $\tau = 12$ sec (Ethereum inter block time).
- $\beta = 2^{40}$ (Set nonce size to 5 bytes). Notice that the nonce range is exhausted by our ASIC in $2^{40}/Q_{\mathsf{asic}} = 0.08$ seconds, so we queue up new work to the ASIC (with a new challenge) every 0.08 seconds.
- $d = 7 \times 10^{13}, n = 12, y = 6$

For this setting, we have completeness probability $> 99.9\%$ and forced revelation error $< 10^{-4}$ against an adversary with a network of 10,000 CPUs.

E.3 Deferred Details on SMACK

Global system architecture. SMACK consists of different verification contracts, one for each type of CK proof method that is supported. For a given public PoK value x, the prover supplies a proof (potentially interactively) according to a certain CK method to the corresponding verification contract. The verification contract stores a boolean mapping from x, indicating whether the proof has been successfully verified. This mapping is leveraged by the CK Registry to provide a uniform and well managed interface to application developers.

CK Registry specification. In Ethereum, public keys are associated with *addresses*. SMACK supports CK for the private keys associated with Ethereum addresses.

CK proofs have an important property: Once one has been generated for a given witness / address, it remains *indefinitely valid*. That is because once a private key sk has been exposed to $\mathcal{E}$, the fact of exposure remains true for all time.

SMACK therefore includes a smart contract, called the *CK Registry*, that maintains a permanent record of addresses for which valid CK proofs have been provided.

The CK Registry (deployed at 0x25B270...eE3966) includes a function that maps Ethereum addresses to the type(s) of CK proofs, if any, that have been verified successfully. When a user wishes to submit a proof of complete knowledge for their Ethereum address, it sends the proof to a verification contract (such as the one we describe in Section 9.7.1) that the CK Registry trusts and then asks the CK Registry to record the event. Applications can then cheaply query the CK Registry to see whether a CK address is verified rather than handling proofs themselves. The CK Registry also allows for extra data to be associated with each verification event (such as attestation from and certificate of a TEE, or the security parameters of the ASIC-ZKPoCK protocol) so that upstream applications can utilize important information about the parameters of verification.

The CK Registry is designed to allow the addition of more CK verification contracts in the future, possibly incorporating new classes of CK proofs as they are designed, and likewise to remove existing CK verification contracts (in case

the parameters are deemed insecure in the future). As a demonstration of our work, we have used the CK Registry and verification contracts to successfully mint Atomic NFTs for CK addresses. The Atomic NFT contract is deployed at 0x6986fc...FcAC38.

Setting the Nonce range β. For exploring the PoW puzzle solutions, the Bitcoin pool mining protocol allows the miners to try different values for the *nonce* and *extranonce* fields. The nonce field in Bitcoin header is fixed to 4 bytes and has proven to be too small for the mining market. Therefore, extranonce was introduced whose size can be set by the pool software. Thus, for our case where the range is $\beta(> 2^{32})$, we set extranonce to $\lfloor log_2[\beta]/8 - 4 \rfloor$ bytes. Note that B in the proof transcript π denotes the portion of the Bitcoin block header that excludes the extranonce field, i.e. we treat that field as part of the space of possible nonces.

APPENDIX F

APPENDIX FOR CHAPTER 10

F.1 Deferred Cryptography Details

Security properties for the computation functionality $\mathcal{F}$. Correctness requires that if all the servers are honest, then the output passes verification. More formally, for all $0 < t < n \in \mathbb{N}$, $(k_1, \ldots, k_n, \mathsf{vk}) \leftarrow \mathcal{F}.\mathsf{Gen}(\lambda, t, n)$, all inputs $x \in \mathcal{X}$, all randomness $r^c \in \mathcal{R}_C$, all subsets $\mathcal{J} \subseteq [n]$ of size t, and all randomness $r_i^s \in \mathcal{R}_S$ for all $i \in \mathcal{J}$, the following holds:

$$\Pr\left[\begin{array}{l|c} \mathcal{F}.\mathsf{Ver}(\mathsf{vk}, x, y, \pi) = 1 \wedge \forall i \in \mathcal{J} : & x^* \leftarrow \mathcal{F}.\mathsf{Query}(x; r^c) \\ \mathcal{F}.\mathsf{PartialVer}(\mathsf{vk}, x^*, y_i^*, \pi_i) = 0 & \forall i \in \mathcal{J} : \\ & (y_i^*, \pi_i) \leftarrow \mathcal{F}.\mathsf{Respond}(k_i, x^*; r_i^s) \\ & (y, \pi) \leftarrow \mathcal{F}.\mathsf{Output}(\{y_i^*\}_{i \in \mathcal{J}}, r_c) \end{array}\right]$$

Further, soundness requires that it should be hard for a client to compute the function at a new input $\hat{x}$. This should hold even if it can see function evaluations at arbitrary inputs of its choice, and upto $t-1$ servers are corrupt. We formalize the notion of a 'new' input based on the leakage—the adversary is not allowed to query partial function evaluations on any input whose leakage belongs in $\mathsf{leak}_{\mathcal{F}}(\hat{x})$[1]. This definition can be seen as a generalization of the unforgeability definition for signatures to arbitrary functionalities. We formalize

[1]Note that this definition is meaningful only in the context of public or one-way function leakages. For oblivious leakage, one could define a variant of soundness similar to the one-more unforgeability property of blind signatures. However, we do not explore this definition in this paper, as our focus is on non-oblivious leakages.

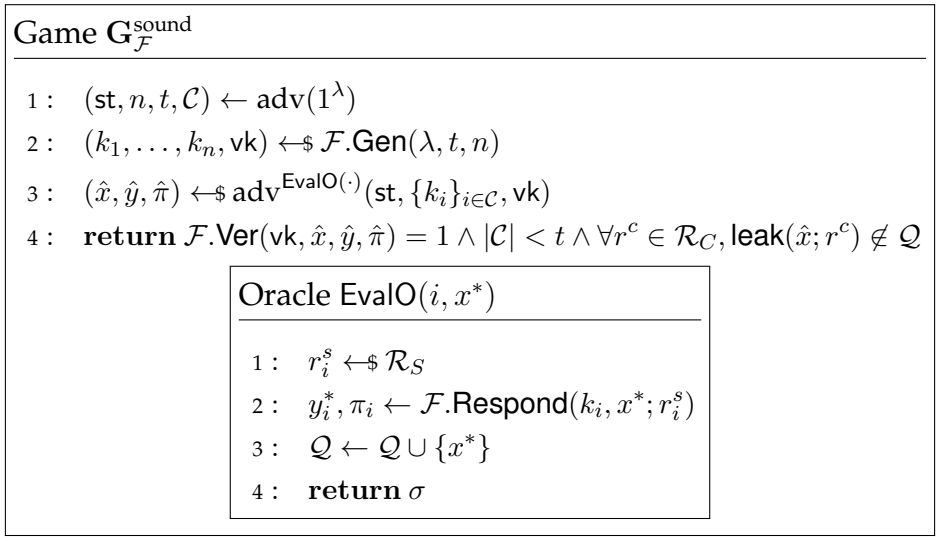

Figure F.1: The soundness game $\mathbf{G}^{\text{sound}}_{\mathcal{F}}$ for a functionality $\mathcal{F}$.

this requirement using a game-based definition, which is presented in Figure F.1 and Definition F.1.

Definition F.1. A functionality $\mathcal{F}$ is sound if, for all $\lambda \in \mathbb{N}$ and all PPT adversaries adv, the following advantage is negligible in λ:

$$\mathbf{Adv}sound\mathcal{F}(\text{adv}) = \Pr[\mathbf{G}^{\text{sound}}_{\mathcal{F}}(\text{adv}) = 1]$$

Formalism for identifiable whistleblowing. We now describe how our model from Section 10.4 can be extended to capture applications such as accountable threshold signatures (e.g. multi-signatures) where it is indeed possible to identify at least one server in the quorum offering off-band service:

- The $\mathcal{F}.\mathsf{Gen}$ algorithm outputs another verification key vk' which can be used to verify proofs that blame certain servers.
- The $\mathcal{F}.\mathsf{Output}$ algorithm now also outputs a set $\mathcal{J}$ of parties being blamed, along with a proof π'.

- We define a new algorithm $\mathcal{F}.\mathsf{VerBlame}$ that takes as input vk', the input-output pair (x, y), the set of servers being blamed $\mathcal{J}$ and the proof π', and outputs a single bit denoting whether all servers in $\mathcal{J}$ were indeed responsible for generating this output.
- We can define a strong unframability property for both the functionality $\mathcal{F}$ and the whistleblowing protocol $\mathcal{W}_{\mathcal{F}}$, which says that even if $n-1$ parties collude, they should not be able to produce a valid proof π' that blames a set $\mathcal{J}^*$ containing an honest party. This can be seen as a generalization of the unforgeability property for multisignatures (see [42] and the references therein).

Example attack if OPRF privacy is relaxed to one-way leakage. OPRFs are widely used in applications such as keyword search, private set intersection, private information retrieval, and password-authenticated key-exchange. As the name suggests, these applications have oblivious leakage. Unfortunately, weakening this to one-way leakage would trivially break these applications. For example, consider Private set intersection (PSI), wherein two parties have sets S_1 and S_2, and the second party P_2 want to compute the intersection $S_1 \cap S_2$ without revealing anything about its set to the first party P_1. Let us assume that the first party has a OPRF key k. It starts with sending the OPRF evaluations $E = \{u_i\}_{i \in S_1}$ at all elements in its set S_1 to the second party P_2. P_2 and P_1 then run the OPRF protocol with P_2 querying P_1 for the evaluation v_j at all points $j \in S_2$. Then, for any $j \in S_2$, if v_j is in E, then P_2 adds j to its output. It is easy to see that P_2 output is equal to $S_1 \cap S_2$. Unfortunately, if P_2 is required to reveal $owf(j)$ for its OPRF queries to P_1, then P_1 can learn a lot about S_2. Specifically, it can learn $S_1 \cap S_2$ by comparing $owf(j)$ to the values $owf(i)$ for all elements i

in its set S_1. A similar attack can be mounted on other applications of OPRFs as well. To conclude, whistleblowing is impossible for such applications, based on Section 10.4.3.1.

F.2 Deferred Cryptography Proofs

F.2.1 Impossibility of Whistleblowing for Oblivious Leakage (Lemma 10.11)

Proof. Suppose that there is a functionality $\mathcal{F}$ (for a function f) with a whistleblowing protocol $(\mathbb{P}, \mathbb{V})$ that is ϵ-sound.

We will now show that the leakage cannot be ϵ-oblivious by constructing a distinguisher $\mathcal{D}$. Let $\mathbf{G}_{ob}$ denote the oblivious leakage game. $\mathcal{D}$ starts with sampling $0 < t < n \in \mathbb{N}$ and $(k_1, \ldots, k_n, \mathsf{vk}) \leftarrow \mathcal{F}.\mathsf{Gen}(\lambda, t, n)$. It then samples $x_0, x_1 \leftarrow\!\!\$\ \mathcal{X}$, sends them to its challenger, and gets back x^*. Additionally, $\mathcal{D}$ runs the $\mathsf{Query}, \mathsf{Respond}$ and Output functions to get a function evaluation at x_0, which we denote by y_0. In other words, $(x, y) \in f_k$.

To distinguish whether x^* came from $\mathsf{leak}(x_0)$ or $\mathsf{leak}(x_1)$, $\mathcal{D}$ simply runs the whistleblowing prover for $(\mathsf{vk}, L = \{x^*\})$ with the witness (x_0, y_0). More formally, it runs $\Pi \leftarrow\!\!\$\ \mathbb{P}((\{x^*\}, \mathsf{vk}), (x_0, y_0))$. If $\mathbb{V}(\{x^*\}, \mathsf{vk}, \Pi)$ outputs 1, then $\mathcal{D}$ guesses 1 (i.e., that x^* came from $\mathsf{leak}(x_1)$); otherwise it guesses 0 (i.e., that x^* came from $\mathsf{leak}(x_0)$).

To compute the advantage of $\mathcal{D}$, let E_0 and E_1 denote the cases $b = 0$ and

$b = 1$ respectively, where b is the bit sampled by the challenger in the oblivious leakage game. First, by completeness of the whistleblowing protocol, we have that for all $L_1 \subseteq \mathsf{leak}(x_1)$:

$$\Pr\left[\mathbb{P}((L_1, \mathsf{vk}), (x_0, y_0)) \leftrightarrow \mathbb{V}(L_1, \mathsf{vk}) \Rightarrow 1\right] = 1.$$

This implies that $\mathcal{D}$ will always output one if $b = 1$, because $\{x^*\}$ is a subset of $\mathsf{leak}(x_1)$ in that case. More formally, $\Pr[\mathbf{G}_{ob}(\mathcal{D}) = 1|\mathsf{E}_1] = 1$.

Next, to analyse the $b = 0$ case, we claim that we can use $\mathcal{D}$ to construct an adversary adv that breaks unframability with advantage equal to $\Pr[\mathbf{G}_{ob}(\mathcal{D}) = 1|\mathsf{E}_0]$. Specifically, adv samples $0 < t < n \in \mathbb{N}$, sends $t, n, \mathcal{C} = \perp$ to its challenger, and receives vk. It then samples x_0, x_1, and $x^* \leftarrow\!\!\$\ \mathsf{leak}(x_0)$. It queries its unframability challenger for t partial evaluations on x^*, for an arbitrary subset of t servers, and combines the outputs to compute y_0. Note that at this point, the leakage set stored by the unframability challenger only includes x^*. Then, similar to $\mathcal{D}$, adv runs the honest prover of the whistleblowing protocol, to get $\Pi^* \leftarrow\!\!\$\ \mathbb{P}((\{x^*\}, \mathsf{vk}), (x_0, y_0))$, and sends Π^* to its challenger.

Now, observe that the distribution of $\mathsf{vk}, x_0, x_1, x^*, y_0$ and Π^* in the above unframability game is identical to that in the oblivious leakage game when $b = 0$. Hence, the probability that adv wins the unframability game is equal to the probability that the verifier $\mathbb{V}$ accepts Π^*, which is $\Pr[\mathbf{G}_{ob}(\mathcal{D}) = 1|\mathsf{E}_0]$. Since the whistleblowing protocol is ϵ-unframeable, this means that $\Pr[\mathbf{G}_{ob}(\mathcal{D}) = 1|\mathsf{E}_0]$ is less than ϵ.

Finally, we compute the advantage of $\mathcal{D}$:

$$\begin{aligned}\mathbf{Adv}ob - leak\mathcal{F}[(\mathcal{D})] &= \left|\frac{1}{2}\cdot(\Pr[\mathbf{G}_{ob}(\mathcal{D})=1|\mathsf{E}_1]+\Pr[\mathbf{G}_{ob}(\mathcal{D})=0|\mathsf{E}_0])-\frac{1}{2}\right| \\ &\geq \left|\frac{1}{2}\cdot(1+1-\epsilon)-\frac{1}{2}\right| \\ &\geq \frac{(1-\epsilon)}{2} \end{aligned} \tag{F.1}$$

This proves that if the whistleblowing protocol is ϵ-unframable, then there exists a distinguisher that breaks oblivious leakage with non-negligible advantage. □

F.2.2 Whistleblowing Protocol for other Leakages

We complete the proof deferred from Section 10.4.3.2.

Completeness. This follows directly from the completeness of the NIZK-PoK system.

Anonymity. This also follows trivially from the honest-verifier zero-knowledge property of the NIZK-PoK system.

Unframability. Theorem F.2 below proves unframability of the above whistleblowing protocol by relying on the knowledge-soundness of the NIZK-PoK proof and the soundness of the functionality $\mathcal{F}$.

Theorem F.2. *For every* PPT *adversary* adv *that breaks unframability of the whistleblowing protocol, there exist another* PPT *adversary* $\mathcal{B}$ *such that,*

$$\mathbf{Adv}unfram\mathcal{W}_{\mathcal{F}}[(\text{adv})] \leq \mathbf{Adv}sound\mathcal{F} + \kappa$$

where κ is the knowledge soundness error of the whistleblowing protocol.

Proof. Given an adversary adv for the unframability game, we will use it to construct an adversary $\mathcal{B}$ for the soundness of the functionality $\mathcal{F}$. Let $\mathsf{E_c}$ denote the event that adv corrupts less than t servers, i.e. $|\mathcal{C}| < t$. Note that adv can win its game only if this event occurs. This implies,

$$\mathbf{Adv}unfram\mathcal{W}_{\mathcal{F}}[(\text{adv})] = \Pr[\mathbf{G}^{\text{unfram}}_{\mathcal{W}_{\mathcal{F}}}(\text{adv}) = 1|\mathsf{E}_c]$$

In other words, we only need to focus on adversaries that corrupt less than t servers.

We now describe how $\mathcal{B}$ works:

- Receive $(n, t, \mathcal{C})$ from adv. Send these to the soundness challenger.
- Receive $\mathsf{vk}, \{k_i\}_{i\in\mathcal{C}}$ from the challenger, and forward to adv.
- For every EvalO oracle query, simply forward it to its challenger, and return its response to adv.

Eventually, adv returns a proof Π. Note that adv can be seen as a NIZK-PoK prover for the statement $(\mathsf{vk}, \mathcal{Q})$, which returns a valid proof with probability $\Pr[\mathbf{G}^{unfram}_{\mathcal{W}_{\mathcal{F}}}(\text{adv}) = 1|\mathsf{E}_c]$. Hence, $\mathcal{B}$ runs the extractor $\mathcal{E}$ on adv to extract the witness $(\hat{x}, \hat{y})$. $\mathcal{B}$ simply outputs $(\hat{x}, \hat{y})$. (Note that there is no $\hat{\pi}$ since we are only considering self-verifiable functionalities.)

Let E_e be the event that the extractor succeeds. Observe that, if this event occurs, then, $(\hat{x}, \hat{y})$ must be a valid witness, which means, $\mathcal{F}.\mathsf{Ver}(\mathsf{vk}, \hat{x}, \hat{y}) = 1$,

and the list $\mathcal{Q}$ does not contain any leakage for $\hat{x}$, i.e. $\forall r^c \in \mathcal{R}_C$, $\mathsf{leak}(\hat{x}; r^c) \notin Q$. This is exactly the winning condition of $\mathcal{B}$. Formally, this means that,

$$\mathbf{Adv}sound\mathcal{F}[(\mathcal{B})] \geq \Pr[\mathsf{E}_e].$$

By knowledge soundness of the whistleblowing protocol, we know that

$$\Pr[\mathsf{E}_e] \geq \Pr[\mathbf{G}^{\text{unfram}}_{\mathcal{W}_\mathcal{F}}(\text{adv}) = 1 | \mathsf{E}_c] - \kappa.$$

This proves the theorem. □

F.3 Existence of a "Reasonable" Second-Stage Pure Nash Equilibrium

In this section, we motivate the assumption in Theorem F.10 on the existence of a second-stage pure Nash equilibria that leaves the coalition with a joint positive utility across the two stages.

All the whistle-blowing mechanisms discussed thus far have a second-stage pure Nash equilibrium. In other words, there exists some second-stage pure Nash equilibrium that the players can coordinate on if they choose to collude, but rather *choose* to remain honest since some player receives a negative utility from colluding. The lower bound in Theorem F.10 considers only such mechanisms with a second-stage pure Nash equilibrium.

On the other hand, we can also construct whistleblowing mechanisms that do not have a second-stage pure Nash equilibrium (Example F.3). In such mech-

anisms, all SPPNE will involve players playing honest in the first stage purely because no second-stage pure Nash equilibrium exists when all players play collude. In other words, the onus of "creating a second-stage pure Nash equilibrium" lies on the retaliation mechanism.

Even though whistleblowing mechanisms with no second-stage pure Nash equilibria deter collusion, from a modelling perspective, such mechanisms should be considered "different" to whistleblowing mechanisms that have a second-stage pure Nash equilibrium.

Example F.3. Consider the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ given by

1. $\mathbf{P}^{\mathbb{W}}(w) = \begin{cases} 81/90 & \text{if } w = 90, \\ 1 & \text{if } w = 100, \\ 0 & \text{if } w = 110, \\ 0 & \text{otherwise} \end{cases}$

   and,

2. $\mathbf{S}^{\mathbb{W}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ 1 & \text{if } w = 90, \\ 15.2 & \text{if } w = 100, \\ 5 & \text{if } w = 110, \\ 15.2 + 1/2^w & \text{otherwise.} \end{cases}$

Proposition F.4. *The whistle-blowing game induced by $\mathcal{M}^{\mathbb{W}}$ does not have a pure Nash equilibrium when the number of players k that know the whistleblowing proof equals 2.*

Proof. It is not hard to see that the two players jointly blowing w whistles for $w \neq 90, 100, 110$ whistles is not an equilibrium. If $w = 0$, then, one of the players

is better off by blowing a 100 whistles. If $w \notin \{0, 90, 100, 110\}$, then, each player is better off blowing 111 more whistles, which reduces the whistleblowing slash without changing the whistleblowing reward.

We now have to rule out equilibria where the players jointly blow $90, 100$ or 110 whistles. First, assume player 1 blows all w whistles. If $w = 90$, then player 1 is better off blowing 100 whistles instead. If $w = 100$, player 2 is better off blowing 10 additional whistles to get slashed 5 instead of getting slashed 15.2 from blowing no whistles. If $w = 110$, player 1 is better off blowing 10 less whistles.

Now, consider players 1 and 2 blowing w_1 and w_2 whistles respectively such that $w = w_1 + w_2$. Without loss of generality, assume $w_1 \geq w_2$.

If $w = 90$, consider player 1 blowing 10 additional whistles. By blowing w_1 whistles, player 1 receives a utility $\frac{81}{90}w_1 - 1$, but by blowing $w_1 + 10$ whistles, it receives a utility $1 \times (w_1 + 10) - 15.2 = w_1 - 4.8$. Note that $w_1 - 4.8 \geq \frac{81}{90}w_1 - 1$ whenever $w_1 \geq 38$. Since, $w_1 + w_2 = 90$ and $w_1 \geq w_2$, w_1 is at least 45, and therefore, player 1 benefits from deviating and blowing 10 additional whistles.

IF $w = 100$, consider player 2 blowing 10 less whistles (we will consider the case where $w_2 < 10$ later). By blowing w_2 whistles, player 2 gets a utility $1 \times w_2 - 15.2$, while by blowing $w_2 - 10$ whistles, it receives a utility $\frac{81}{90}(w_2 - 10) - 1 = \frac{81}{90}w_2 - 10$. The latter is greater than the former whenever $w_2 \leq 52$. Indeed, $w_1 + w_2 = 100$ and $w_2 \leq w_1$, and thus, $w_2 < 52$. Therefore, player 2 is better off deviating and blowing 10 less whistles.

If $w = 100$ and $w_2 < 10$, then player 2 is better off blowing 10 more whistles. By blowing w_2 whistles, it receives a utility $w_2 - 15.2 < 10 - 15.2 = -5.2$. By

blowing 10 more whistles, it receives a utility $0 - 5 = -5$, which is strictly better.

Finally, if $w = 110$, player 1 is better of blowing 10 less whistles to receive a reward $(w_1 - 10) - 15.2 > -5$, the reward from blowing 110 whistles in total.

Thus, $\mathbb{W}$ does not have a pure Nash equilibrium. □

It might also be the case that all pure Nash equilibria induced by $\mathcal{M}^{\mathbb{W}}$ can be extremely "unrealistic." For example, in r-whistle mechanisms, every player blowing $n \times r$ whistles is a pure Nash equilibria for any $k > 2$. Even if some player deviates and blows a different number of whistles, the total number of whistles blown is strictly greater than r and thus, all players receive a zero reward from the whistleblowing mechanism and are slashed their entire deposit. However, one can claim that whistleblowing mechanisms with such "unrealistic" equilibria should "spiritually" be considered similarly as whistleblowing mechanisms with no second-stage pure Nash equilibria. Assuming that the coalition gets a positive joint utility in the pure Nash equilibrium across the two stages necessarily ensures that the equilibrium is "interesting."

Theorem F.10 argues that the 1-whistle $(n-1)\,\gamma$-secure mechanism is optimal amongst all whistleblowing mechanisms that support a second stage pure Nash equilibrium where the coalition earns a net positive utility across the two stages of $\mathbb{W}$.

However, there can also exist whistleblowing mechanisms with "interesting" second stage equilibria that results in the joint net utility of the coalition in $\mathbb{W}$ being negative.

Example F.5. Consider the 2-whistle mechanism $\mathcal{M}^{\mathbb{W}}$ given by

1. $\mathbf{P}^{\mathbb{W}}(w) = \begin{cases} V + n\gamma + \varepsilon & \text{if } w = 1, \\ \varepsilon & \text{if } w = 2, \\ 0 & \text{otherwise} \end{cases}$

   and,

2. $\mathbf{S}^{\mathbb{W}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ V + \gamma + \frac{\varepsilon}{2} & \text{otherwise.} \end{cases}$

Note that blowing 0 or 1 whistle is not an equilibrium when $k \geq 2$. Any of the k players blowing zero whistles is incentivized to blow a whistle in both cases. Further, note that 2 players blowing a whistle each is an equilibrium. Neither of the whistleblowers will want to blow a different number of whistles (the whistleblowing slash remains the same, but the reward decreases from ε to zero if they blow a different number of whistles), while the other players are indifferent between blowing no whistles and blowing some number of whistles.

The joint utility of the coalition across the two stages, given by $nV - n \times (V + \gamma + \frac{\varepsilon}{2}) + 2\varepsilon$ is strictly negative. However, the Nash equilibrium where two players blow a whistle each is certainly "interesting," unlike the equilibrium where every player blows much more than r whistles in a r-whistle mechanism.

We leave the problem of classifying all whistleblowing mechanisms with "interesting" pure Nash equilibria and finding the optimal whistleblowing mechanism (i.e, the mechanism $\mathcal{M}^{\mathbb{W}}$ with a given whistleblowing deposit $D^{\mathbb{W}}$ that requires the largest retaliation deposit to be broken) open.

F.4 Alternate Models for the Whistleblowing and Retaliation Games

In this section, we explore various other models for the whistleblowing and retaliation games that could arise naturally in applications.

F.4.1 Automated Redirection of Whistleblowing Rewards

In the discussions thus far, we have implicitly assumed that the players blow whistles from privately controlled accounts to which the whistleblowing rewards are transferred (the *private accounts model*). Thus, the deposit collected by the retaliation mechanism must also be large enough so that the whistleblowers do not "abscond" after getting rewarded by the whistleblowing mechanism without sharing it with the remainder of the players. We can also consider alternate collusion models, where the players collude by blowing whistles through jointly controlled wallets. Since the whistleblowing rewards are transferred to the joint wallets, the retaliation mechanism is not responsible to prevent the whistleblowers from absconding. We present two possible models for whistleblowing and retaliation from joint accounts. In the first model, some player will still have to submit a whistleblowing proof on behalf of the joint account, while in the second, blowing whistles from joint accounts is fully automated.

Retaliation with jointly controlled wallets but non-automated whistleblowing. We start by describing the model with jointly controlled accounts where the players still have to blow whistles on behalf of the account — we call this

the *non-automated joint accounts model*. While the first round is identical to the private accounts model, the action-space for the players $1, \ldots, k$ in the second round consists of $(u_i, v_i) \in (\mathbb{N} \cup \{0\})^2$. u_i denotes the number of whistles blown by player i from its private accounts while v_i denotes the number of whistles it blows on behalf of joint accounts.

Each player then derives a utility $u_i \mathbf{P}^{\mathbb{W}}(\sum_i u_i + v_i) - \mathbf{S}^{\mathbb{W}}(\sum_i u_i + v_i)$ from the whistleblowing mechanism and an additional utility equal to $\mathcal{R}_i(\sum_i u_i + v_i)$ from the retaliation mechanism.[2] Since the retaliation mechanism collects some of the whistleblowing rewards and distributes it back to the players, budget balance for $\mathcal{R}$ would now require $\sum_i \mathcal{R}(v) \leq v\mathbf{P}^{\mathbb{W}}(v)$ for all $v \in \mathbb{N} \cup \{0\}$. In other words, the net reward distributed by $\mathcal{R}$ cannot be larger than the total money it receives from the whistleblowing mechanism. The non-automated joint accounts model is otherwise identical to the private accounts model.

It should generally be possible to stop players from blowing whistles from jointly controlled accounts. Using such joint accounts effectively amounts to whistleblowing through smart contract addresses instead of fresh privately-owned addresses (e.g., EOAs on Ethereum). However, since it is typically visible if the calling entity is an EOA or a smart contract, a whistleblowing mechanism may choose to only allow EOAs to blow whistles.

Further, even if joint accounts arise through private means (e.g., a trusted hardware that controls the key), we note that if desired, the whistleblowing protocol designer could prevent joint accounts through recent cryptographic

[2] To be precise, the retaliation mechanism $\mathcal{R}_i$ should take as input the total number of whistles $\sum_i u_i + v_i$ blown by the players (which can be learnt by $\mathcal{R}$ since $\mathcal{M}^{\mathbb{W}}$ publishes this information anyways) and the joint accounts which received a reward from $\mathcal{M}^{\mathbb{W}}$. However, for notational cleanliness, we do not explicitly mention the joint accounts as input to $\mathcal{R}$, but they can be deduced from the pair (u_i, v_i) played by player i.

techniques [156, 107] that allow proving *individual* ownership of keys. This also prevents automated whistleblowing from joint wallets.

Even if the protocol designer cannot contain coalitions from using joint accounts, we will argue that breaking a whistleblowing mechanism is equally difficult with purely private accounts and with jointly controlled accounts as long as whistleblowing cannot be automated.

Theorem F.6. *A whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ can be broken by a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with retaliation deposit $D^{\mathbb{R}}$ and some SPPNE $\vec{w}^2$ in the private accounts model if and only if there exists a retaliation mechanism $\mathcal{R}$ in the non-automated joint accounts model with the same retaliation deposit $D^{\mathbb{R}}$ such that $\mathcal{R}$ and some SPPNE $((u_1, v_1), \ldots, (u_n, v_n))^2$ breaks $\mathcal{M}^{\mathbb{W}}$.*

Proof. Proving the "only if" direction is easy — if there exists a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ and $\vec{w}^2$ that breaks $\mathcal{M}^{\mathbb{W}}$ purely with private accounts, then $\mathcal{M}^{\mathbb{R}}$ and $\vec{w}^2$ also breaks $\mathcal{M}^{\mathbb{W}}$ in the non-automated joint accounts model. Formally, $\mathcal{R}_i(w) = -\mathcal{M}_i^{\mathbb{R}}(w)$.

We use the following intuition to prove the "if" direction. While the retaliation mechanism gets direct control of the funds flowing into the jointly controlled accounts, instead of blowing whistles from the joint accounts of the coalition, players can potentially blow whistles from their private accounts and then abscond with all the whistleblowing rewards without sharing them. For instance, if the coalition strategically decides that player i must blow u_i whistles from private accounts and v_i whistles from joint accounts, player i might instead blow $w_i = u_i + v_i$ whistles from its private accounts and pocket all of the whistleblowing rewards that would have otherwise flown into the joint accounts. We will argue that stopping the above "redirection" attack requires as

much collateral as stopping the player from absconding in the private accounts model.

Consider a retaliation mechanism $\mathcal{R}$ and SPPNE $((u_1, v_1), \ldots, (u_n, v_n))^2$ that breaks $\mathcal{M}^{\mathbb{W}}$ in the joint accounts model. We will show a reduction to construct a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ in the private accounts model that also breaks $\mathcal{M}^{\mathbb{W}}$. Instead of first handing over the whistleblowing rewards to the coalition and then receiving some share of the rewards via $\mathcal{R}$, we will construct $\mathcal{M}^{\mathbb{R}}$ such that each player directly receives the rewards from $\mathcal{M}^{\mathbb{W}}$ and then forwards the money to be redistributed. Define $\mathcal{M}^{\mathbb{R}}$ to be

$$\mathcal{M}_i^{\mathbb{R}}(\sum_j w_j) = \begin{cases} v_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathcal{R}_i(\sum_j w_j) & \text{if } \sum_j w_j = \sum_j u_j + v_j, \\ D^{\mathbb{R}} & \text{otherwise.} \end{cases}$$

We argue that $(w_1, \ldots, w_n)^2$ is a SPPNE in the retaliation game $\mathbb{R}$ where $w_i = u_i + v_i$ for all $1 \leq i \leq n$.

To this extent, we will begin by showing that $(w_1, \ldots, w_n)$ is a second-stage Nash equilibrium in the private accounts model. Further, when all players $j \neq i$ blow w_j whistles, we argue that player i's utility from blowing w_i whistles under the retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ equals the utility from blowing u_i private whistles and v_i whistles from joint accounts under $\mathcal{R}$. Player i's utility from

blowing w_i whistles equals

$$
\begin{aligned}
& w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) - \mathcal{M}_i^{\mathbb{R}}(\sum_j w_j) \\
&= (u_i + v_i)\mathbf{P}^{\mathbb{W}}(\sum_j u_j + v_j) - \mathbf{S}^{\mathbb{W}}(\sum_j u_j + v_j) - \mathcal{M}_i^{\mathbb{R}}(\sum_j u_j + v_j) \\
&= u_i \mathbf{P}^{\mathbb{W}}(\sum_j u_j + v_j) - \mathbf{S}^{\mathbb{W}}(\sum_j u_j + v_j) \\
&\qquad\qquad + (v_i \mathbf{P}^{\mathbb{W}}(\sum_j u_j + v_j) - \mathcal{M}_i^{\mathbb{R}}(\sum_j u_j + v_j)) \\
&= u_i \mathbf{P}^{\mathbb{W}}(\sum_j u_j + v_j) - \mathbf{S}^{\mathbb{W}}(\sum_j u_j + v_j) + \mathcal{R}_i(\sum_i u_i + v_i),
\end{aligned}
$$

which equals the utility from blowing u_i private whistles and v_i whistles on behalf of joint accounts. On the other hand, by deviating and blowing $w_i' \neq w_i$ whistles in the private accounts model, player i receives a second-stage utility

$$
\begin{aligned}
& w_i' \mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - D^{\mathbb{R}} \\
&\leq w_i' \mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j \neq i} u_j + v_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j \neq i} u_j + v_j) \\
&\qquad\qquad + \mathcal{R}_i(w_i' + \sum_{j \neq i} u_j + v_j) \\
&\leq u_i \mathbf{P}^{\mathbb{W}}(u_i + v_i + \sum_{j \neq i} u_j + v_j) - \mathbf{S}^{\mathbb{W}}(u_i + v_i + \sum_{j \neq i} u_j + v_j) \\
&\qquad\qquad + \mathcal{R}_i(u_i + v_i + \sum_{j \neq i} u_j + v_j) \\
&= w_i \mathbf{P}^{\mathbb{W}}(w_i + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i + \sum_{j \neq i} w_j) - \mathcal{M}_i^{\mathbb{R}}(w_i + \sum_{j \neq i} w_j)
\end{aligned}
$$

The second line follows since $\mathcal{R}$ cannot penalize player i more than $D^{\mathbb{R}}$. The third line follows since blowing u_i private whistles and v_i whistles from joint accounts is a Nash equilibrium in the non-automated joint accounts model — the left hand side is the utility from blowing w_i' private whistles while the right

hand side is the equilibrium utility. The final equality follows from the definition of $\mathcal{M}^{\mathbb{R}}$.

To conclude the proof, we need to argue that $(w_1, \ldots, w_n)^2$ satisfies Equation (10.4) for all players i (i.e, each player earns a net positive utility across the two rounds) and is therefore, a SPPNE. However, it immediately follows since the players receive a utility identical to $((u_1, v_1), \ldots, (u_n, v_n))^2$, which satisfies Equation (10.4). Thus, $(w_1, \ldots, w_n)^2$ is indeed a SPPNE and hence, $\mathcal{M}^{\mathbb{R}}$ breaks $\mathcal{M}^{\mathbb{W}}$. □

Retaliation with jointly controlled wallets and automated whistleblowing. The reduction in Theorem F.6 that proved breaking a whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ in the private accounts model and the non-automated joint accounts model are equally easy (or difficult) crucially uses the fact that players can deviate from the coalition's strategy to not blow whistles on behalf of the joint accounts, but rather blow whistles from accounts that are their own. To circumvent such deviations by the players, the coalition can look to automate whistleblowing from joint accounts (call this the *automated joint accounts model*). In the automated joint accounts model, players are still free to blow whistles from their private accounts. However, they will not be able to stop whistles from getting submitted on behalf of the accounts jointly controlled by the coalition (if the coalition decides to submit one).

Formally, automated whistleblowing modifies the retaliation game as follows. The first round is identical to the two models discussed thus far — players either collude to receive a utility V each or remain honest. In the second round however, the coalition submits v whistles from jointly controlled accounts for

some v determined before the start of the first round (thus, the players cannot tamper with these v whistles). Player i's action in the second round, as in the private accounts model, consists of blowing $w_i \in \mathbb{N} \cup \{0\}$ whistles from its privately owned accounts. For a retaliation mechanism $\mathcal{R}$, player i receives a utility $w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j + v) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j + v)$ from the whistleblowing mechanism and $\mathcal{R}_i(\sum_j w_j + v)$ from the retaliation mechanism. The model is otherwise identical to the non-automated joint accounts model.

We will denote the second-stage actions taken by the players and the coalition in the retaliation game $\mathbb{R}$ by $((w_1, \ldots, w_n), v)$. Similar to the private accounts model, the strategy profile $(\mathsf{collude}, \ldots, \mathsf{collude}), ((w_1, \ldots, w_n), v)$ will then be denoted by $((w_1, \ldots, w_n), v)^{\mathbb{2}}$.

We argue that it is strictly harder to prevent players from colluding in the automated joint accounts model than in the private accounts model (and therefore, the non-automated joint accounts model). In particular, we will sketch a retaliation mechanism that does not collect any deposit from the players, but still breaks the 1-whistle skeleton mechanism which was sufficient to prevent collusion in the private accounts model.

Proposition F.7. *There exists a retaliation mechanism $\mathcal{R}$ that collects no deposits such that $\mathcal{R}$ and $((0, \ldots, 0), 1)^{\mathbb{2}}$ breaks the 1-whistle skeleton mechanism.*

Proof. Recall that the 1-whistle skeleton disburses a reward $V + \varepsilon$ in the case of a sole whistleblower, but never rewards whistleblowers otherwise. It penalizes all players $V + \frac{\varepsilon}{n-1}$ even if a single whistle is blown.

Consider the retaliation mechanism that sets up exactly 1 joint account and blows a whistle on behalf of this account. Further, it distributes $\frac{V+\varepsilon}{n}$ to each of

the n players if and only if no other whistle is blown.

In the second round, observe that none of the players are incentivized to blow any whistles — on top of the $V + \frac{\varepsilon}{n-1}$ slashed by the whistleblowing mechanism, the players receive $\frac{V+\varepsilon}{n}$ from blowing no whistles, but a zero utility from blowing a non-zero number of whistles. Thus, $((0, \ldots, 0), 1)$ is a second-stage equilibrium. We can quickly verify that each player receives a utility $\frac{V}{n} - \varepsilon(\frac{1}{n-1} - \frac{1}{n}) > 0$ across the two stages of $\mathbb{R}$ by playing $((0, \ldots, 0), 1)^2$, which is better than the utility from staying honest. Thus, $((0, \ldots, 0), 1)^2$ is a SPPNE and the retaliation mechanism discussed above and $((0, \ldots, 0), 1)^2$ breaks the 1-whistle skeleton mechanism. □

However, the 1-whistle $(n-1)\,\gamma$-secure mechanism can be modified to collect a slightly larger deposit to be secure against retaliation mechanisms with $D^{\mathbb{R}} \leq (n-1)\,\gamma$ that use automated whistles. Consider the following "aggressive" 1-whistle $(n-1)\,\gamma$-secure mechanism:

1. $\mathbf{P}^{\mathbb{W}}(w) = \begin{cases} \frac{n}{n-1}V + n\,\gamma + \varepsilon & \text{if } w = 1, \\ 0 & \text{otherwise} \end{cases}$
   for some small $\varepsilon > 0$, and,
2. $\mathbf{S}^{\mathbb{W}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ \frac{n}{n-1}V + \gamma + \frac{\varepsilon}{n-1} & \text{otherwise.} \end{cases}$

Note that $\mathcal{M}^{\mathbb{W}}$ collects a deposit $D^{\mathbb{W}} = \frac{n}{n-1}V + \gamma + \frac{\varepsilon}{n-1}$.

Proposition F.8. *The aggressive 1-whistle $(n-1)\,\gamma$-secure mechanism cannot be broken by any retaliation mechanism that collects a deposit of at most $(n-1)\,\gamma$ from each of its players, irrespective of the number of players k that learn the whistleblowing proof.*

Unlike the proof approach sketched in Section 10.6.3, we run a top-down dynamic program to argue Proposition F.8. We start by arguing that no SPPNE consists of one or more whistles getting blown. We then rule out a second-stage Nash equilibrium with no whistles.

Proof. Assume for contradiction that $\vec{w}^2$ is a SPPNE in the retaliation game $\mathbb{R}$ induced by some retaliation mechanism $\mathcal{R}$, where $\vec{w}$ is a pure Nash equilibrium consisting of one or more whistles blown. We will argue that the coalition jointly receives a negative utility, and thus, some player receives a negative utility from colluding in the first stage and will deviate to play honest.

If one or more whistles are blown in the second phase, each player is slashed $\frac{n}{n-1}V + \gamma + \frac{\varepsilon}{n-1}$ for a total slash of $\frac{n^2}{n-1}V + n\gamma + \frac{n}{n-1}\varepsilon$. The whistleblowers, on the other hand, are rewarded at most $\frac{n}{n-1}V + n\,\gamma + \varepsilon$ in total. Thus, the net utility of the coalition across the two rounds can be upper bounded by

$$nV + \left(\frac{n}{n-1}V + n\,\gamma + \varepsilon\right) - \left(\frac{n^2}{n-1}V + n\gamma + \frac{n}{n-1}\varepsilon\right) < 0.$$

The first term (nV) is the collusion utility from the first round. Thus, the coalition achieves a negative utility by colluding and blowing one or more whistles.

Finally, we will argue that blowing no whistles cannot be a second-stage equilibrium for any retaliation mechanism $\mathcal{R}$ with $D^{\mathbb{R}} \leq (n-1)\,\gamma$. If no whistles are blown, some player i strictly profits by deviating and blowing a whistle.

$$\begin{aligned}
&1 \times \mathbf{P}^{\mathbb{W}}(1) - \mathbf{S}^{\mathbb{W}}(1) + \mathcal{R}_i(1) \\
&\quad \geq \left(\frac{n}{n-1}V + n\,\gamma + \varepsilon\right) - \left(\frac{n}{n-1}V + \gamma + \frac{\varepsilon}{n-1}\right) - (n-1)\,\gamma \\
&\quad > 0.
\end{aligned}$$

Thus, blowing no whistles cannot be a second-stage equilibrium.

Hence, no retaliation mechanism with a deposit $(n-1)\,\gamma$ can break the aggressive 1-whistle $(n-1)\,\gamma$-secure mechanism in the joint accounts model with automated whistleblowing. □

F.4.2 Anonymous vs Non-Anonymous Whistleblowing Models

We have assumed anonymous whistleblowing in all the models considered so far — i.e, the identities of the whistleblowers remain unknown to both the whistleblowing mechanism and the retaliation mechanism. In this section, we consider a relaxed non-anonymous model of whistleblowing where the coalition learns exactly how many whistles were blown by each player. In other words, the retaliation mechanism can depend on the number of whistles blown by each of the n players.

We will argue that non-anonymity will not help coalitions in designing better retaliation mechanisms — for a given whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$, if there exists a retaliation mechanism and a SPPNE that breaks $\mathcal{M}^{\mathbb{W}}$ in the non-anonymous model, then, there exists a retaliation mechanism with the same deposit that breaks $\mathcal{M}^{\mathbb{W}}$ in the anonymous model.

Theorem F.9. *Suppose $\mathcal{M}^{\mathbb{W}}$ is broken by a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ that learns the number of whistles blown by each player and a SPPNE $(w_1, \ldots, w_n)^2$. Then, there exists an anonymous retaliation mechanism $\hat{\mathcal{M}}^{\mathbb{R}}$ that collects the same retaliation deposit as $\mathcal{M}^{\mathbb{R}}$ such that $\hat{\mathcal{M}}^{\mathbb{R}}$ and $(w_1, \ldots, w_n)^2$ breaks $\mathcal{M}^{\mathbb{W}}$.*

Proof. At a high level, the retaliation mechanism expects the players to coordinate on the second-stage equilibrium $(w_1, \ldots, w_n)$ and thus, expects $\sum_i w_i$ whis-

tles to be blown in total. If a different number of whistles are submitted to $\mathcal{M}^{\mathbb{W}}$, then the retaliation mechanism does not have to investigate which player i deviated from blowing w_i whistles — it can penalize everybody.

Construct $\hat{\mathcal{M}}^{\mathbb{R}}$ as follows.

$$\hat{\mathcal{M}}_i^{\mathbb{R}}(w) = \begin{cases} \mathcal{M}_i^{\mathbb{R}}(w_1, \ldots, w_n) & \text{if } w = \sum_i w_i, \\ D^{\mathbb{R}} & \text{otherwise.} \end{cases}$$

It is not hard to see that $\hat{\mathcal{M}}^{\mathbb{R}}$ does not need access to the exact number of whistles blown by each of the n players. It only needs the aggregate number of whistles w blown to execute, which is published by $\mathcal{M}^{\mathbb{W}}$. Further, it also requires only the same retaliation deposit as $\mathcal{M}^{\mathbb{R}}$.

We begin by arguing that $(w_1, \ldots, w_n)$ is a second-stage equilibrium (Equation (10.3)) in the retaliation game induced by $\hat{\mathcal{M}}_i^{\mathbb{R}}$. Observe that

$$\begin{aligned}
& w_i \mathbf{P}^{\mathbb{W}}(w_i + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i + \sum_{j \neq i} w_j) - \hat{\mathcal{M}}^{\mathbb{R}}(w_i + \sum_{j \neq i} w_j) \\
&= w_i \mathbf{P}^{\mathbb{W}}(w_i + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i + \sum_{j \neq i} w_j) - \mathcal{M}^{\mathbb{R}}(w_1, \ldots, w_n) \\
&\geq w_i' \mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \mathcal{M}^{\mathbb{R}}(w_i', w_{-i}) \\
&\geq w_i' \mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - D^{\mathbb{R}} \\
&= w_i' \mathbf{P}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \mathbf{S}^{\mathbb{W}}(w_i' + \sum_{j \neq i} w_j) - \hat{\mathcal{M}}^{\mathbb{R}}(w_i' + \sum_{j \neq i} w_j).
\end{aligned}$$

Here (w_i, w_{-i}) is the vector obtained by replacing w_i by w_i' in $\vec{w}$. The inequality in the third line follows since $\vec{w}$ is a second-stage equilibrium in $\mathbb{R}$ induced by $\mathcal{M}^{\mathbb{R}}$. The fourth line follows since $\mathcal{M}^{\mathbb{R}}(\cdot) \leq D^{\mathbb{R}}$. The final equality follows from the construction of $\hat{\mathcal{M}}^{\mathbb{R}}$. Thus, $(w_1, \ldots, w_n)$ is a second-stage equilibrium in the retaliation game induced by $\hat{\mathcal{M}}^{\mathbb{R}}$.

We can immediately conclude that $\vec{w}^2$ is a SPPNE is the retaliation game induced by $\hat{\mathcal{M}}^{\mathbb{R}}$. When each player i blows w_i whistles, their net utility across the two rounds is identical between the two retaliation mechanisms $\mathcal{M}^{\mathbb{R}}$ and $\hat{\mathcal{M}}^{\mathbb{R}}$. Thus, if all players receive a positive utility under $\mathcal{M}^{\mathbb{R}}$ (since $\vec{w}$ is a SPPNE under $\mathcal{M}^{\mathbb{R}}$), then, all players also receive a net positive utility under $\hat{\mathcal{M}}^{\mathbb{R}}$. Thus, $\vec{w}^2$ is a SPPNE.

We have therefore proved that $\hat{\mathcal{M}}^{\mathbb{R}}$ and $\vec{w}^2$ breaks $\mathcal{M}^{\mathbb{W}}$. □

F.5 Deferred Game-Theoretic Proofs

F.5.1 Proof of Theorem 10.16

For a given whistleblowing deposit $D^{\mathbb{W}}$, we will argue that any whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ can be broken by a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with a sufficiently large deposit $D^{\mathbb{R}}$. Intuitively, if the retaliation mechanism collects a deposit much larger than the utility from collusion and the whistleblowing rewards, $\mathcal{M}^{\mathbb{R}}$ can threaten to burn the entire $D_i^{\mathbb{R}}$ for every player i even if a single whistle is blown. Players are better off receiving zero utility in the second stage from blowing no whistles than incurring the massive retaliation penalty from blowing whistles.

Formally, consider the retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ that collects a deposit $D_i^{\mathbb{R}} = (n-1) \times D^{\mathbb{W}}$ from each player i given by

$$\mathcal{M}_i^{\mathbb{R}}(w) = \begin{cases} 0 & \text{if } w = 0 \\ D^{\mathbb{R}} & \text{otherwise.} \end{cases}$$

We will argue that $(0,\ldots,0)^2$ is a SPPNE of $\mathbb{R}$. To see Equation (10.3) is satisfied for all $1 \leq i \leq k$, observe that for $w \geq 1$

$$\begin{aligned}
&w_i \mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) - \mathcal{M}_i^{\mathbb{R}}(w) \\
&\leq w\mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) - \mathcal{M}_i^{\mathbb{R}}(w) \\
&\leq n\mathbf{S}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) - \mathcal{M}_i^{\mathbb{R}}(w) \\
&\leq (n-1) \times D^{\mathbb{W}} - (n-1) \times D^{\mathbb{W}} \leq 0.
\end{aligned}$$

The third line follows from budget balance of $\mathcal{M}^{\mathbb{W}}$. Thus, the players are all better off not blowing any whistles in the second stage. Indeed, Equation (10.4) is trivially satisfied. All players receive a net utility equal to V across the two stages playing $(0,\ldots,0)^2$, which is strictly better than the utility from being honest. Thus, $(0,\ldots,0)^2$ is a SPPNE of $\mathbb{R}$ and $\mathcal{M}^{\mathbb{R}}$ and $(0,\ldots,0)^2$ breaks $\mathcal{M}^{\mathbb{W}}$.

F.5.2 Proof of Theorem 10.17

Consider any retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with a retaliation deposit $D^{\mathbb{R}} \leq (n-1)\,\gamma$. We will calculate the bottom-up dynamic program described in Section 10.6.3 to calculate all SPPNE of the induced retaliation game $\mathbb{R}$.

First, observe that any pure Nash equilibria in the second round consists of some player blowing at least one whistle. Indeed, if nobody blows any whistles, player i receives a zero second-round utility. On the other hand, if i deviates and blows a whistle, it is rewarded $nV + n\gamma + \varepsilon$, slashed $V + \gamma + \frac{\varepsilon}{n-1}$ by $\mathcal{M}^{\mathbb{W}}$ and is penalized at most $(n-1)\,\gamma$ by $\mathcal{M}^{\mathbb{R}}$, which yields a utility at least $(n-1)V + (1 - \frac{1}{n-1})\,\varepsilon > 0$. Thus, blowing no whistles is not a second-stage equilibrium.

Next, we observe that if some whistle is blown, then, there exists some player

who is better off playing honest in the first round than colluding. If two or more whistles are blown, then, every player is slashed at least $V + \gamma$ and is not rewarded at all by $\mathcal{M}^{\mathbb{W}}$. Even with the collusion reward V, all of them end up with a negative net utility. On the other hand, suppose that exactly one whistle is blown. We show that the $(n-1)$ players that did not blow the whistle will end up with a negative joint utility, and thus, at least one of them must receive a negative utility across the two rounds. The $(n-1)$ players together receive a collusion reward equal to $(n-1)V$ in the first stage and are slashed $(n-1)\left(V+\gamma+\frac{\varepsilon}{n-1}\right)$ in the second by the whistleblowing mechanism. They are awarded at most $(n-1)\gamma$ by the retaliation mechanism, leaving them with a joint utility of at most $-\varepsilon < 0$, as required.

Thus, no retaliation mechanism with a deposit at most $(n-1)\gamma$ can break the 1-whistle $(n-1)\gamma$-secure mechanism.

F.5.3 Proof of Theorem 10.18

We will argue that the retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ described below and some $\vec{w}^2$ will break any whistleblowing mechanism with deposit up to $V + \gamma$. $\mathcal{M}^{\mathbb{R}}$ penalizes player 1 nothing if no whistles are blown, and collects $(n-1)\gamma$ from player 1 and distributes γ each to the remaining $(n-1)$ players even if a single whistle is blown.

1. $\mathcal{M}_1^{\mathbb{R}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ (n-1)\gamma & \text{otherwise.} \end{cases}$

2. $\mathcal{M}_i^{\mathbb{R}}(w) = \begin{cases} 0 & \text{if } w = 0, \\ -\gamma & \text{otherwise,} \end{cases}$
for $i \neq 1$.

We will analyze two separate cases — the largest utility $\max_w w\mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w)$ that player 1 can derive from the whistleblowing mechanism is (a) smaller than $(n-1)\,\gamma$ and (b) larger than $(n-1)\,\gamma$.

In the former case, $\mathcal{M}^{\mathbb{R}}$ will disincentivize player 1 from blowing any whistles. The whistleblowing reward it receives is smaller than the retaliation penalty. Formally, suppose that $w\mathbf{S}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) \leq 0 = 0 \times \mathbf{P}^{\mathbb{W}}(0) - \mathbf{S}^{\mathbb{W}}(0)$ for all $w \in \mathbb{N}$. Then, blowing zero whistles is a second-stage Nash equilibrium. Indeed, all of the n players will receive a net utility V across the two stages. Thus, both Equation (10.3) and Equation (10.4) are satisfied for all $1 \leq i \leq n$, and $(0, \ldots, 0)^2$ is a subgame perfect Nash equilibrium.

In the latter case, player 1 will blow whistles to optimize its second-stage utility and share $(n-1)\,\gamma$ of the whistleblowing rewards with the other $(n-1)$ players. Suppose that $w = \arg\max_{\hat{w}} \hat{w}\mathbf{P}^{\mathbb{W}}(\hat{w}) - \mathbf{S}^{\mathbb{W}}(\hat{w})$ and $w\mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) \geq (n-1)\,\gamma$. Then, blowing w whistles is a second-stage pure Nash equilibrium (Equation (10.3) is satisfied for player 1). To verify that Equation (10.4) is satisfied, note that all players $i \neq 1$ are slashed $\mathbf{S}^{\mathbb{W}}(w) \leq V + \gamma$ by the whistleblowing mechanism and are allocated γ by the retaliation mechanism. Thus, their net utility across both the rounds

$$\begin{aligned} U_i^{\mathbb{R}}((w, 0 \ldots, 0)^2) &= V - \mathbf{S}^{\mathbb{W}}(w) + \mathcal{M}_i^{\mathbb{R}}(w) \\ &\leq V - (V + \gamma) + \gamma \\ &= 0. \end{aligned}$$

Player 1 on the other hand receives a utility $w\mathbf{P}^{\mathbb{W}}(w) - \mathbf{S}^{\mathbb{W}}(w) \geq (n-1)\,\gamma$ from $\mathcal{M}^{\mathbb{W}}$ and is charged $(n-1)\,\gamma$ by the retaliation mechanism. Thus, it receives a positive second-stage utility and therefore a positive utility across the two stages.

Thus, $\mathcal{M}^{\mathbb{R}}$ breaks any whistleblowing mechanism that collects a deposit at most $V + \gamma$ for some strategy profile $\vec{w}^2$.

F.5.4 Optimality of $\mathbb{M}^{\mathbb{W}}_{1*}$

Theorem F.10. *Let $\mathcal{M}^{\mathbb{W}}$ be a whistleblowing mechanism with a second-stage pure Nash equilibrium $\vec{w} = (w_1, \ldots, w_n)$ such that the joint utility $\sum_i U_i^{\mathbb{R}}((w_1, \ldots, w_n)^2) = \sum_i \left(V + w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j)\right)$ of the n players across the two rounds is strictly positive. If $\mathcal{M}^{\mathbb{W}}$ collects a whistleblowing deposit $V + \gamma$, then there exists $\mathcal{M}^{\mathbb{R}}$ that collects a deposit $(n-1)\,\gamma$ such that $\mathcal{M}^{\mathbb{R}}$ and $(w_1, \ldots, w_n)^2$ breaks $\mathcal{M}^{\mathbb{W}}$.*

Proof. At a high level, we will consider a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ that reallocates utility from each player that receives a positive utility in the equilibrium $\vec{w} = (w_1, \ldots, w_n)$ (call them *receivers*) to the remaining players (call them *defaulters*) irrespective of the number of whistles blown. Since the utilities of the players are shifted by the same constant irrespective of their second-round actions, $\vec{w}$ remains an equilibrium. We have to ensure that the redistribution occurs so that each player ends up with a positive net utility $U_i^{\mathbb{R}}(\vec{w}^2)$ across the two stages.

Consider the retaliation mechanism that collects $\min\{(n-1)\,\gamma, w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) + V\}$ from each receiver i and distributes $\max\{\mathbf{S}^{\mathbb{W}}(\sum_j w_j) - w_{i'} \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - V, 0\}$ to each defaulter i', irrespective of the number of whistles

blown.

It is fairly straightforward to argue that $\vec{w}^2$ satisfies Equation (10.3) and Equation (10.4) for all players i. Each receiver is penalized at most an additional V on top of their whistleblowing reward $w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j)$, and thus combined with the collusion reward V in the first round, their net utility across the rounds is non-negative. Similarly, each defaulter is compensated at least V less than what they lose in the whistleblowing mechanism, and thus, they receive a net non-negative utility combined with the collusion reward V. Hence, $\vec{w}^2$ is a SPPNE in $\mathbb{R}$.

It is easy to verify that $\mathcal{M}^{\mathbb{R}}$ collects a deposit at most $(n-1)\,\gamma$. Indeed, only the receivers are charged a positive penalty by $\mathcal{M}^{\mathbb{R}}$ and are charged at most $(n-1)\,\gamma$. Thus, $D^{\mathbb{R}} \leq (n-1)\,\gamma$.

It only remains to verify budget balance of $\mathcal{M}^{\mathbb{R}}$, i.e, that the net penalty charged by $\mathcal{M}^{\mathbb{R}}$ is non-negative. We consider two cases for the same — (a) all receivers are charged $w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) + V$ and (b) some receiver is charged a penalty $(n-1)\,\gamma$. In the first case, the total penalty charged by $\mathcal{M}^{\mathbb{R}}$ from both receivers and defaulters is bounded below by

$$\sum_i \left(w_i \mathbf{P}^{\mathbb{W}}(\sum_j w_j) - \mathbf{S}^{\mathbb{W}}(\sum_j w_j) + V \right) \geq 0.$$

The inequality follows since the left hand side is the joint net utility of the coalition across the two stages in the whistleblowing game $\mathbb{W}$ for the SPPNE $\vec{w}^2$, which is larger than zero, as assumed in the proposition statement. In the second case when some receiver i is penalized $(n-1)\,\gamma$, note that no defaulter is

rewarded more than γ by $\mathcal{M}^{\mathbb{R}}$.

$$\begin{aligned}\mathcal{M}_{i'}^{\mathbb{R}}(w) &\leq \mathbf{S}^{\mathbb{W}}(\sum_j w_j) - w_{i'}\mathbf{P}^{\mathbb{W}}(\sum_j w_j) - V \\ &\leq \mathbf{S}^{\mathbb{W}}(\sum_j w_j) - V \\ &\leq (V+\gamma) - V \leq \gamma.\end{aligned}$$

The last line follows since the whistleblowing deposit $D^{\mathbb{W}} \leq V + \gamma$. It immediately follows that the $(n-1)\,\gamma$ penalty collected from receiver i is sufficient to cover the compensations rolled out by $\mathcal{M}^{\mathbb{R}}$ to all the defaulters. Thus, $\mathcal{M}^{\mathbb{R}}$ satisfies budget balance as required.

To summarize, $\mathcal{M}^{\mathbb{R}}$ with $D^{\mathbb{R}} \leq (n-1)\,\gamma$ and $\vec{w}^2$ breaks the whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ with $D^{\mathbb{W}} \leq V + \gamma$. □

F.5.5 Deferred details for (t, n)-collusion

Below, we state the lower bound result for whistleblowing in the (t, n)-setting. The proof proceeds analogously to the lower bound proofs in the (n, n) setting.

Proposition F.11. *1. For any whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ in the (t, n)-collusion environment which collects a deposit $D^{\mathbb{W}}$, there exists a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with a retaliation deposit $D^{\mathbb{R}} = (n-1) \times D^{\mathbb{W}}$ such that $\mathcal{M}^{\mathbb{R}}$ and some SPPNE $\vec{w}^2$ breaks $\mathcal{M}^{\mathbb{W}}$.*

2. *Further, if the number of players k that know the whistleblowing proof equals 1, then any whistleblowing mechanism $\mathcal{M}^{\mathbb{W}}$ with a whistleblowing deposit $D^{\mathbb{W}} = V + \gamma$ can be broken by some retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ and a SPPNE $\vec{w}^2$, where $D^{\mathbb{R}} = (t-1)\,\gamma$.*

3. *Finally, if there exists a second-stage Nash equilibrium in the whistleblowing game $\mathbb{W}$ induced by $\mathcal{M}^{\mathbb{W}}$ with $D^{\mathbb{W}} = V + \gamma$ such that the joint utility of the coalition of t players across the two stages is positive, then, there exists a retaliation mechanism $\mathcal{M}^{\mathbb{R}}$ with deposit $D^{\mathbb{R}} = (t-1)\,\gamma$ and some SPPNE that breaks $\mathcal{M}^{\mathbb{W}}$.*

BIBLIOGRAPHY

[1] Ittai Abraham, Lorenzo Alvisi, and Joseph Y Halpern. Distributed computing meets game theory: combining insights from two fields. *ACM Sigact News*, 42(2):69–76, 2011.

[2] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Alexander Spiegelman. Solida: A blockchain protocol based on reconfigurable Byzantine consensus. In *OPODIS*, pages 25:1–25:19, 2017.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *IEEE S&P*, pages 106–118, 2020.

[4] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous Byzantine agreement. In *PODC*, pages 337–346, 2019.

[5] Ittai Abraham, Benny Pinkas, and Avishay Yanai. Blinder – scalable, robust anonymous committed broadcast. In *CCS*, page 1233–1252, 2020.

[6] Amazon Web Services. Aws nitro enclaves website. `https://aws.amazon.com/ec2/nitro/nitro-enclaves/`, Referenced Nov. 2022.

[7] AMD. AMD SEV-SNP: Strengthening VM isolation with integrity protection and more. *White Paper*, Jan. 2020.

[8] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE TDSC*, 8(4):564–577, 2011.

[9] Prabhanjan Ananth, Gilad Asharov, Hila Dahari, and Vipul Goyal. Towards accountability in CRS generation. In *Eurocrypt*, pages 278–308, 2021.

[10] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. Innovative technology for CPU based attestation and sealing. In *HASP*, page 7, 2013.

[11] Android Developers: Verifying hardware-backed key pairs with key attestation. `https://developer.android.com/training/articles/security-key-attestation#root_certificate`, Referenced Nov. 2022.

[12] Android Open Source Project. Android Open Source Project: Key and id attestation. https://source.android.com/docs/security/features/keystore/attestation, Referenced Nov. 2022.

[13] Apple Inc. Apple developer website: Establishing your app's integrity. https://developer.apple.com/documentation/devicecheck/establishing_your_app_s_integrity, Referenced Nov. 2022.

[14] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. A fair consensus protocol for transaction ordering. In *ICNP*, pages 55–65, 2018.

[15] Gilad Asharov, Ran Canetti, and Carmit Hazay. Towards a game theoretic view of secure computation. In *Eurocrypt*, pages 426–445, 2011.

[16] Giuseppe Ateniese, Jan Camenisch, Marc Joye, and Gene Tsudik. A practical and provably secure coalition-resistant group signature scheme. In *CRYPTO*, pages 255–270, 2000.

[17] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant Byzantine fault tolerance. In *ICDCS*, pages 297–306, 2013.

[18] James Austgen, Andrés Fábrega, Sarah Allen, Kushal Babel, Mahimna Kelkar, and Ari Juels. DAO decentralization: Voting-bloc entropy, bribery, and Dark DAOs, 2023. https://arxiv.org/abs/2311.03530.

[19] James Austgen, Andrés Fábrega, Mahimna Kelkar, Dani Vilardell, Sarah Allen, Kushal Babel, Jay Yu, and Ari Juels. Liquefaction: Privately liquefying blockchain assets. In *IEEE S&P*, 2025.

[20] Kushal Babel and Lucas Baker. Strategic peer selection using transaction value and latency. In *DeFi @ CCS*, pages 9–14, 2022.

[21] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts. In *IEEE S&P*, pages 2499–2516, 2023.

[22] Vivek Kumar Bagaria, Sreeram Kannan, David Tse, Giulia C. Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *CCS*, pages 585–602, 2019.

[23] Leemon Baird. The Swirlds Hashgraph consensus algorithm: Fair, fast, Byzantine fault tolerance, 2016. https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf.

[24] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. SoK: Consensus in the age of blockchains. In *AFT*, pages 183–198, 2019.

[25] Salvador Barbera and Matthew O Jackson. Strategy-proof exchange. *Econometrica*, 63(1):51–87, 1995.

[26] Mathieu Baudet, George Danezis, and Alberto Sonnino. Fastpay: High-performance byzantine fault tolerant settlement. In *ACM AFT*, pages 163–177, 2020.

[27] Joseph Bebel and Dev Ojha. Ferveo: Threshold decryption for mempool privacy in BFT networks. Cryptology ePrint Archive, Paper 2022/898, 2022.

[28] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[29] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections. In *STOC*, pages 544–553, 1994.

[30] B Douglas Bernheim, Bezalel Peleg, and Michael D Whinston. Coalition-proof Nash equilibria i. concepts. *Journal of economic theory*, 42(1):1–12, 1987.

[31] Alysson Bessani, João Sousa, and Eduardo E.P. Alchieri. State machine replication for the masses with bft-smart. In *DSN*, pages 355–362, 2014.

[32] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *ITCS*, page 326–349, 2012.

[33] Bitmain. Specifications of T19/S19 liquid-cooling miner. https://support.bitmain.com/hc/en-us/articles/4418373232153, 11 Feb. 2022.

[34] BlockPI, Accessed 2023. https://blockpi.io/.

[35] Remco Bloemen, Leonid Logvinov, and Jacob Evans. EIP 712. `https://eips.ethereum.org/EIPS/eip-712`, 2017.

[36] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *CRYPTO*, pages 757–788, 2018.

[37] Dan Boneh and Chelsea Komlo. Threshold signatures with private accountability. In *CRYPTO*, pages 551–581, 2022.

[38] Dan Boneh, Aditi Partap, and Lior Rotem. Accountability for misbehavior in threshold decryption via threshold traitor tracing. In *CRYPTO*, pages 317–351, 2024.

[39] Dan Boneh, Aditi Partap, and Lior Rotem. Traceable secret sharing: Strong security and efficient constructions. In *CRYPTO*, pages 221–256, 2024.

[40] Dan Boneh, Aditi Partap, and Lior Rotem. Proactive refresh for accountable threshold signatures. In *FC*, pages 140–159, 2025.

[41] Dan Boneh, Aditi Partap, and Lior Rotem. Traceable verifiable random functions. Cryptology ePrint Archive, Paper 2025/312, 2025.

[42] Dan Boneh, Aditi Partap, and Brent Waters. Accountable multi-signatures with constant size public keys. Cryptology ePrint Archive, Paper 2023/1793, 2023.

[43] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use PGP. In *WPES*, pages 77–84, 2004.

[44] Pietro Borrello, Andreas Kogler, Martin Schwarzl, Moritz Lipp, Daniel Gruss, and Michael Schwarz. ÆPIC leak: Architecturally leaking uninitialized data from the microarchitecture. In *USENIX Security*, pages 3917–3934, 2022.

[45] Gabriel Bracha and Sam Toueg. Asynchronous consensus and broadcast protocols. *J. ACM*, 32(4):824–840, 1985.

[46] John Brainard, Ari Juels, Ronald L Rivest, Michael Szydlo, and Moti Yung. Fourth-factor authentication: somebody you know. In *CCS*, pages 168–178, 2006.

[47] Lorenz Breidenbach, Phil Daian, Ari Juels, and Florian Tramèr. To sink frontrunners, send in the submarines, 2017. https://hackingdistributed.com/2017/08/28/submarine-sends/.

[48] Eric Budish, Andrew Lewis-Pye, and Tim Roughgarden. The economic limits of permissionless consensus. In *EC*, pages 704–731, 2024.

[49] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE S&P*, pages 315–334, 2018.

[50] Benedikt Bünz and Ben Fisch. Transparent snarks from dark compilers. In *EUROCRYPT*, pages 677–706, 2020.

[51] V. Buterin. Why we need wide adoption of social recovery wallets. vitalik.ca blog post at https://vitalik.ca/general/2021/01/11/recovery.html, 11 Jan 2021.

[52] V. Buterin. Minimal anti-collusion infrastructure (MACI). Ethereum Research blog post at https://ethresear.ch/t/minimal-anti-collusion-infrastructure/5413, 2 May 2019.

[53] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

[54] Christian Cachin, Jovana Micic, Nathalie Steinhauer, and Luca Zanolini. Quick order fairness, 2021. arXiv:2112.06615.

[55] Christian Cachin and Marko Vukolić. Blockchain consensus protocols in the wild, 2017. arXiv:/1707.01873.

[56] Paul Camion. Chemins et circuits hamiltoniens des graphes complets. *Comptes Rendus de l'Académie des Sciences de Paris*, 249:2151–2152, 1959.

[57] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–147, 2001.

[58] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[59] Ran Canetti, Yevgeniy Dodis, Rafael Pass, and Shabsi Walfish. Universally composable security with global setup. In *TCC*, pages 61–85, 2007.

[60] Ran Canetti, Kyle Hogan, Aanchal Malhotra, and Mayank Varia. A universally composable treatment of network time. In *CSF*, pages 360–375, 2017.

[61] Ran Canetti and Tal Rabin. Universal composition with joint state. In *CRYPTO*, pages 265–281, 2003.

[62] Georg Cantor. Beiträge zur begründung der transfiniten mengenlehre. *Mathematische Annalen*, pages 481–512, 1895.

[63] Sílvia Casacuberta, Julia Hesse, and Anja Lehmann. Sok: Oblivious pseudorandom functions. In *EuroS&P*, pages 625–646, 2022.

[64] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, pages 173–186, 1999.

[65] Chainlink. Smartcon 2022 research and product development highlights, 2022. `https://blog.chain.link/smartcon-research-updates/`.

[66] Christopher P Chambers and Federico Echenique. *Revealed preference theory*, volume 56. Cambridge University Press, 2016.

[67] T-H. Hubert Chan, Naomi Ephraim, Antonio Marcedone, Andrew Morgan, Rafael Pass, and Elaine Shi. Blockchain with varying number of players. Cryptology ePrint Archive, Report 2020/677, 2020. `https://eprint.iacr.org/2020/677`.

[68] T.-H. Hubert Chan, Rafael Pass, and Elaine Shi. Consensus through herding. In *EUROCRYPT*, pages 720–749, 2019.

[69] David Chaum. Blind signatures for untraceable payments. In *CRYPTO*, pages 199–203, 1983.

[70] David Chaum, Peter YA Ryan, and Steve Schneider. A practical voter-verifiable election scheme. In *ESORICS*, pages 118–139, 2005.

[71] David L Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981.

[72] Jing Chen and Silvio Micali. Collusive dominant-strategy truthfulness. *Journal of Economic Theory*, 147(3):1300–1312, 2012.

[73] Wuhui Chen, Yikai Feng, Jianting Zhang, Zhongteng Cai, Hong-Ning Dai, and Zibin Zheng. Auncel: Fair byzantine consensus protocol with high performance. In *INFOCOM*, pages 1251–1260, 2024.

[74] James Hsin-yu Chiang, Bernardo David, Tore Kasper Frederiksen, Arup Mondal, and Esra Yeniaras. Detecting rogue decryption in (threshold) encryption via self-incriminating proofs. Cryptology ePrint Archive, Paper 2024/794, 2024.

[75] Tarun Chitra, Matheus V. X. Ferreira, and Kshitij Kulkarni. Credible, optimal auctions via public broadcast. In *AFT*, volume 316 of *LIPIcs*, pages 19:1–19:16, 2024.

[76] Kevin Choi, Aathira Manoj, and Joseph Bonneau. SoK: Distributed randomness beacons. In *IEEE S&P*, pages 75–92, 2023.

[77] Benny Chor, Amos Fiat, and Moni Naor. Tracing traitors. In *CRYPTO*, pages 257–270, 1994.

[78] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private information retrieval. In *FOCS*, pages 41–50, 1995.

[79] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. Fluid mpc: Secure multiparty computation with dynamic participants. In *CRYPTO*, pages 94–123, 2021.

[80] Hao Chung and Elaine Shi. Foundations of transaction fee mechanism design. In *SODA*, pages 3856–3899, 2023.

[81] Michele Ciampi, Aggelos Kiayias, and Yu Shen. Universal composable transaction serialization with order fairness. In *CRYPTO*, pages 147–180, 2024.

[82] Jeremy Clark and Urs Hengartner. Selections: Internet voting with over-the-shoulder coercion-resistance. In *FC*, pages 47–61, 2011.

[83] Michael R Clarkson, Stephen Chong, and Andrew C Myers. Civitas: Toward a secure voting system. In *IEEE S&P*, pages 354–368, 2008.

[84] Allen Clement, Edmund Wong, Lorenzo Alvisi, and Mike Dahlin. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *NDSI*, pages 153–168, 2009.

[85] Shir Cohen, Neel Basu, Soumya Basu, and Lorenzo Alvisi. On fair ordering and differential privacy, 2025. https://arxiv.org/abs/2501.05535.

[86] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the Signal messaging protocol. In *EuroS&P*, pages 451–466, 2017.

[87] Coindesk. Ethermine adds front-running software to help miners offset EIP 1559 revenue losses, 2021.

[88] Condorcet paradox, Accessed 2023. https://wikipedia.org/wiki/Condorcet_paradox.

[89] Thomas Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[90] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. In *Eurocrypt*, pages 103–118, 1997.

[91] Flaviu Cristian, Houtan Aghili, Ray H Strong, and Danny Dolev. Atomic broadcast: From simple message diffusion to Byzantine agreement. *Information and Computation*, 118(1):158–179, 1995.

[92] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *IEEE S&P*, pages 585–602, 2020.

[93] Philip Daian, Tyler Kell, Ian Miers, and Ari Juels. On-chain vote buying and the rise of Dark DAOs. https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/, 2018.

[94] Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. Distributed randomness using weighted VUFs. Cryptology ePrint Archive, Paper 2024/198, 2024.

[95] Alex Davidson, Ian Goldberg, Nick Sullivan, George Tankersley, and Filippo Valsorda. Privacy pass: Bypassing internet challenges anonymously. *Proceedings on Privacy Enhancing Technologies*, 2018:164–180, 06 2018.

[96] Alfredo De Santis, Yvo Desmedt, Yair Frankel, and Moti Yung. How to share a function securely. In *STOC*, 1994.

[97] Henry de Valence. The Penumbra protocol. `https://protocol.penumbra.zone/main/crypto/flow-encryption/threshold-encryption.html`.

[98] Gerard Debreu. Representation of a preference ordering by a numerical function. *Decision processes*, 3:159–165, 1954.

[99] Alan Deckelbaum and Silvio Micali. Collusion, efficiency, and dominant strategies. *Games and Economic Behavior*, 103:83–93, 2017.

[100] Nikhil R Devanur, Yuval Peres, and Balasubramanian Sivan. Perfect bayesian equilibria in repeated sales. In *SODA*, pages 983–1002, 2014.

[101] Mario Di Raimondo, Rosario Gennaro, and Hugo Krawczyk. Deniable authentication and key exchange. In *CCS*, pages 400–409, 2006.

[102] Jack Doerner, Yashvanth Kondi, Eysa Lee, Abhi Shelat, and LaKyah Tyner. Threshold BBS+ signatures for distributed anonymous credential issuance. In *IEEE S&P*, pages 773–789, 2023.

[103] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput*, 12:656–666, 1983.

[104] drand Team. drand: Cryptography, 2025. `https://drand.love/docs/cryptography/`.

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

[106] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, pages 139–147, 1992.

[107] Stefan Dziembowski, Sebastian Faust, and Tomasz Lizurej. Individual cryptography. In *CRYPTO*, pages 547–579, 2023.

[108] Stefan Dziembowski, Sebastian Faust, Tomasz Lizurej, and Marcin Mielniczuk. Secret sharing with snitching. In *CCS*, page 840–853, 2024.

[109] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. SoK: Transparent dishonesty: Front-running attacks on blockchain. In *FC*, pages 170–189, 2019.

[110] Meryem Essaidi, Matheus VX Ferreira, and S Matthew Weinberg. Credible, strategyproof, optimal, and bounded expected-round single-item auctions for all distributions. *arXiv preprint arXiv:2205.14758*, 2022.

[111] Ethereum. `https://ethereum.org/`.

[112] Ethereum consensus notes. `https://en.bitcoin.it/wiki/Stratum_mining_protocol`, [Accessed December 2022].

[113] Ethereum consensus notes. `https://eth2book.info/bellatrix/part2/building_blocks/randomness/`, [Accessed December 2022].

[114] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *FC*, pages 436–454, 2014.

[115] Guy Fayolle, Vadim Aleksandrovich Malyshev, and Mikhail Menshikov. *Topics in the constructive theory of countable Markov chains*. Cambridge university press, 1995.

[116] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of cryptology*, 1(2):77–94, 1988.

[117] Joan Feigenbaum, Christos Papadimitriou, and Scott Shenker. Sharing the cost of muliticast transmissions (preliminary version). In *STOC*, pages 218–227, 2000.

[118] Ed Felton. Mev auctions considered harmful, 2020. `https://medium.com/offchainlabs/mev-auctions-considered-harmful-fa72f61a40ea`.

[119] O. Fernau. Royalty-free sudoswap is finding favor with NFT traders. *The Defiant*, 13 Aug. 2022.

[120] Matheus VX Ferreira, Yotam Gafni, and Max Resnick. Incentive-

compatible collusion-resistance via posted prices. *arXiv preprint arXiv:2412.20853*, 2024.

[121] Matheus VX Ferreira and S Matthew Weinberg. Credible, truthful, and two-round (optimal) auctions via cryptographic commitments. In *EC*, pages 683–712, 2020.

[122] Marc Fischlin. Communication-efficient non-interactive proofs of knowledge with online extractors. In *CRYPTO*, pages 152–168, 2005.

[123] Flashbots. Flashbots: Frontrunning the mev crisis, 2021. `https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752`.

[124] Georg Fuchsbauer and Mathias Wolf. Concurrently secure blind schnorr signatures. In *Eurocrypt*, pages 124–160, 2024.

[125] Aadityan Ganesh, Clayton Thomas, and S Matthew Weinberg. Revisiting the primitives of transaction fee mechanism design. *arXiv preprint arXiv:2410.07566*, 2024.

[126] Juan Garay, Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *FOCS*, pages 648–657, 2013.

[127] Juan Garay and Aggelos Kiayias. Sok: A consensus taxonomy in the blockchain era. In *CT-RSA*, pages 284–318, 2020.

[128] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol with chains of variable difficulty. In *CRYPTO*, pages 291–323, 2017.

[129] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, pages 281–310, 2015.

[130] Sanjam Garg, Abhishek Jain, Pratyay Mukherjee, Rohit Sinha, Mingyuan Wang, and Yinuo Zhang. Cryptography with weights: Mpc, encryption and signatures. In *CRYPTO*, pages 295–327, 2023.

[131] Ivan Geffner and Moshe Tennenholtz. Strengthening Nash equilibria. *arXiv preprint arXiv:2312.14745*, 2023.

[132] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling Byzantine agreements for cryptocurrencies. In *SOSP*, pages 51–68, 2017.

[133] Andrew V Goldberg and Jason D Hartline. Collusion-resistant mechanisms for single-parameter agents. In *SODA*, pages 620–629, 2005.

[134] Sharon Goldberg, Moni Naor, Dimitrios Papadopoulos, Leonid Reyzin, and Sachin Vasantand Asaf Ziv. Nsec5: Provably preventing dnssec zone enumeration. In *NDSS*, 2015.

[135] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, pages 218–229, 1987.

[136] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *STOC*, pages 218–229, 1987.

[137] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. In *STOC*, pages 291–304, 1985.

[138] Tiantian Gong, Aniket Kate, Hemanta K. Maji, and Hai H. Nguyen. Disincentivize collusion in verifiable secret sharing. Cryptology ePrint Archive, Paper 2025/446, 2025.

[139] Vipul Goyal, Yifan Song, and Akshayaram Srinivasan. Traceable secret sharing and applications. In *CRYPTO*, pages 718–747, 2021.

[140] Vincent Gramoli, Zhenliang Lu, Qiang Tang, and Pouriya Zarbafian. AOAB: Optimal and fair ordering of financial transactions. In *DSN*, pages 377–388, 2024.

[141] Jerry Green and Jean-Jacques Laffont. On coalition incentive compatibility. *The Review of Economic Studies*, 46(2):243–254, 1979.

[142] Adam Groce, Jonathan Katz, Aishwarya Thiruvengadam, and Vassilis Zikas. Byzantine agreement with a rational adversary. In *ICALP*, pages 561–572, 2012.

[143] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT*, pages 305–326, 2016.

[144] Lachlan J Gunn, Ricardo Vieitez Parra, and N Asokan. Circumventing cryptographic deniability with remote attestation. *PETS*, 2019(3):350–369, 2019.

[145] Joseph Halpern and Vanessa Teague. Rational secret sharing and multiparty computation: extended abstract. In *STOC*, page 623–632, 2004.

[146] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[147] Mor Harchol-Balter. *Performance modeling and design of computer systems: queueing theory in action*. Cambridge University Press, 2013.

[148] Chi Ho, Danny Dolev, and Robbert van Renesse. Making distributed systems robust. In *OPODIS*, pages 232–246, 2007.

[149] Nicole Immorlica, Brendan Lucier, Emmanouil Pountourakis, and Samuel Taggart. Repeated sales with multiple strategic buyers. In *EC*, pages 167–168, 2017.

[150] Intel Enhanced Privacy ID (EPID) Security Technology. `sintel.com/content/www/us/en/developer/articles/technical/intel-enhanced-privacy-id-epid-security-technology.html`, 2022.

[151] Intel software guard extensions. `https://software.intel.com/en-us/sgx`.

[152] Markus Jakobsson and Ari Juels. Proofs of work and bread pudding protocols. In *Secure information networks*, pages 258–272, 1999.

[153] K. Wei Jie. Release announcement: MACI 1.0. Medium Post at `https://medium.com/privacy-scaling-explorations/release-announcement-maci-1-0-c032bddd2157`, 12 Oct. 2021.

[154] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *WPES*, pages 61–70, 2005.

[155] Ghassan O. Karame, Elli Androulaki, and Srdjan Capkun. Double spending fast payments on bitcoin. In *CCS*, pages 906–917, 2012.

[156] Mahimna Kelkar, Kushal Babel, Philip Daian, James Austgen, Vitalik Buterin, and Ari Juels. Complete knowledge: Preventing encumbrance of cryptographic secrets. In *CCS*, pages 2415–2429, 2024.

[157] Mahimna Kelkar, Soubhik Deb, and Sreeram Kannan. Order-fair consensus in the permissionless setting. In *APKC*, pages 3–14, 2022.

[158] Mahimna Kelkar, Soubhik Deb, Sishan Long, Ari Juels, and Sreeram Kannan. Themis: Fast, strong order-fairness in byzantine consensus. In *CCS*, pages 475–489, 2023.

[159] Mahimna Kelkar, Aadityan Ganesh, Aditi Partap, Joseph Bonneau, and S. Matthew Weinberg. Breaking omertà: On threshold cryptography, smart collusion, and whistleblowing, 2025.

[160] Mahimna Kelkar, Yunqi Li, Nerla Jean-Louis, Carolina Ortega Pérez, Kushal Babel, Andrew Miller, and Ari Juels. The sting framework: Proving the existence of superclass adversaries. Cryptology ePrint Archive, Paper 2024/1676, 2024.

[161] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for Byzantine consensus. In *CRYPTO*, pages 451–480, 2020.

[162] Aggelos Kiayias and Philip Lazos. SoK: Blockchain governance. *arXiv preprint 2201.07188*, 2022.

[163] Aggelos Kiayias, Nikos Leonardos, and Yu Shen. Ordering transactions with bounded unfairness: Definitions, complexity and constructions. In *EUROCRYPT*, pages 34–63, 2024.

[164] Aggelos Kiayias, Alexander Russel, David Bernardo, and Roman Oliynykov. Ouroboros: A provably secure proof of stake blockchain protocol. In *CRYPTO*, pages 357–388, 2017.

[165] Aggelos Kiayias and Qiang Tang. How to keep a secret: Leakage deterring public-key cryptosystems. In *CCS*, pages 943–954, 2013.

[166] Ariah Klages-Mundt and Andreea Minca. (in)stability for the blockchain: Deleveraging spirals and stablecoin attacks, 2020. arXiv:1906.02152.

[167] Eleftherios Kokoris-Kogias, Philipp Eleftherios, Linus Gasser, Nicolas

Gailly, Ewa Syta, and Brian Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE S&P*, pages 583–598, 2018.

[168] Yashvanth Kondi and abhi shelat. Improved straight-line extraction in the random oracle model with applications to signature aggregation. In *ASIACRYPT*, 2022.

[169] Vijay Krishna. *Auction Theory*. Academic Press, 2002.

[170] Klaus Kursawe. Wendy, the good little fairness widget: Achieving order fairness for blockchains. In *ACM AFT*, pages 25–36, 2020.

[171] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *TOPLAS*, 4(3):382–401, 1982.

[172] Kfir Lev-Ari, Alexander Spiegelman, Idit Keidar, and Dahlia Malkhi. Fairledger: A fair blockchain protocol for financial institutions. In *OPODIS*, pages 1:1–1:16, 2019.

[173] Michael Lewis. *Flash Boys: A Wall Street Revolt*. WW Norton & Company, 2014.

[174] libhotstuff: A general-purpose BFT state machine replication library with modularity and simplicity, 2018. `https://github.com/hot-stuff/libhotstuff`.

[175] Brad Linder. Some apps may stop working on rooted Android phones due to SafetyNet update, 11 Mar. 2020.

[176] Wouter Lueks, Iñigo Querejeta-Azurmendi, and Carmela Troncoso. VoteAgain: A scalable coercion-resistant voting system. In *USENIX Security*, pages 1553–1570, 2020.

[177] Loi Luu, Yaron Velner, Jason Teutsch, and Prateek Saxena. Smartpool: Practical decentralized pooled mining. In *USENIX Security*, pages 1409–1426, 2017.

[178] Dahlia Malkhi and Pawel Szalachowski. Maximal extractable value (mev) protection on a dag, 2022.

[179] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn.

Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *CCS*, page 2111–2128, 2019.

[180] Akaki Mamageishvili, Mahimna Kelkar, Jan Christoph Schlegel, and Edward W. Felten. Buying time: Latency racing vs. bidding for transaction ordering. In *AFT*, pages 23:1–23:22, 2023.

[181] Y. Manoussakis. A linear-time algorithm for finding hamiltonian cycles in tournaments. *Discrete Appl. Math.*, 36(2):199–201, 1992.

[182] Alex Manuskin. The fastest draw on the blockchain: Ethereum backrunning, 2020. `https://amanusk.medium.com/the-fastest-draw-on-the-blockchain-bzrx-example-6bd19fabdbe1`.

[183] Jean-Philippe Martin and Lorenzo Alvisi. Fast Byzantine consensus. *IEEE TDSC*, 3(3):202–215, 2006.

[184] Eric Maskin and John Riley. Uniqueness of equilibrium in sealed high-bid auctions. *Games and Economic Behavior*, 45(2):395–409, 2003.

[185] Patrick McCorry, Alexander Hicks, and Sarah Meiklejohn. Smart contracts for bribing miners. Cryptology ePrint Archive, Report 2018/581, 2018. `https://eprint.iacr.org/2018/581`.

[186] Patrick McCorry, Siamak F Shahandashti, and Feng Hao. A smart contract for boardroom voting with maximum voter privacy. In *FC*, pages 357–375, 2017.

[187] Frank McKeen, Ilya Alexandrovich, Ittai Anati, Dror Caspi, Simon Johnson, Rebekah Leslie-Hurd, and Carlos Rozas. Intel® software guard extensions (Intel® SGX) support for dynamic memory management inside an enclave. In *HASP*, pages 1–9, 2016.

[188] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. Innovative instructions and software model for isolated execution. In *HASP*, page 10, 2013.

[189] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. CONIKS: Bringing key transparency to end users. In *USENIX Security*, pages 383–398, 2015.

[190] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *FOCS*, pages 120–130, 1999.

[191] Andrew Miller, Ahmed Kosba, Jonathan Katz, and Elaine Shi. Nonoutsourceable scratch-off puzzles to discourage bitcoin mining coalitions. In *ACM CCS*, pages 680–691, 2015.

[192] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In *ACM CCS*, pages 31–42, 2016.

[193] Ciamac C. Moallemi and Mehmet Saglam. OR forum - the cost of latency in high-frequency trading. *Oper. Res.*, 61(5):1070–1086, 2013.

[194] Hervé Moulin. Incremental cost sharing: Characterization by coalition strategy-proofness. *Social Choice and Welfare*, 16(2):279–320, 1999.

[195] Roger B. Myerson. Optimal auction design. *Math. Oper. Res.*, 6(1):58–73, 1981.

[196] Heena Nagda, Shubhendra Pal Singhal, Mohammad Javad Amiri, and Boon Thau Loo. Rashnu: Data-dependent order-fairness. *Proc. VLDB Endow.*, 17(9):2335–2348, 2024.

[197] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. `https://bitcoin.org/bitcoin.pdf`.

[198] Douglass C. North. Institutions and credible commitment. *Journal of Institutional and Theoretical Economics (JITE) / Zeitschrift für die gesamte Staatswissenschaft*, 149(1):11–23, 1993.

[199] Oasis Labs website. `https://www.oasislabs.com/`, [Accessed June 2022].

[200] Alex Obadia. Quantifying mev: Introducing mev-explore v0, 2021. `https://medium.com/flashbots/quantifying-mev-introducing-mev-explore-v0-5ccbee0f6d02`.

[201] Osmosis docs. `https://docs.osmosis.zone/overview`.

[202] Rafael Pass. On deniability in the common reference string and random oracle model. In *CRYPTO*, pages 316–337, 2003.

[203] Rafael Pass, Lior Seerman, and abhi shelat. Analysis of the blockchain protocol in asynchronous networks. In *EUROCRYPT*, pages 643–673, 2017.

[204] Rafael Pass and Elaine Shi. Fruitchains: A fair blockchain. In *PODC*, pages 315–324, 2017.

[205] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, pages 1–16, 2017.

[206] Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, volume 91, pages 39:1–39:16, 2017.

[207] Rafael Pass and Elaine Shi. Rethinking large-scale consensus. In *CSF*, pages 115–129, 2017.

[208] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *EUROCRYPT*, pages 3–33, 2018.

[209] Rafael Pass, Elaine Shi, and Florian Tramèr. Formal abstractions for attested execution secure processors. In *EUROCRYPT*, pages 260–289, 2017.

[210] Rafael Nat Josef Pass. *A precise computational approach to knowledge*. PhD thesis, Massachusetts Institute of Technology, 2006.

[211] Michal Penn, Maria Polukarov, and Moshe Tennenholtz. Congestion games with failures. In *EC*, pages 259–268, 2005.

[212] Hoai Luan Pham, Thi Hong Tran, Tri Dung Phan, Vu Trung Duong Le, Duc Khai Lam, and Yasuhiko Nakashima. Double SHA-256 hardware architecture with compact message expander for bitcoin mining. *IEEE Access*, 8:139634–139646, 2020.

[213] Pocket network, Accessed 2023. `https://www.pokt.network/`.

[214] David Pointcheval and Jacques Stern. Security arguments for digital signatures and blind signatures. *Journal of cryptology*, 13(3):361–396, 2000.

[215] Ivan Puddu, Daniele Lain, Moritz Schneider, Elizaveta Tretiakova, Sinisa Matetic, and Srdjan Capkun. TEEvil: Identity lease via trusted execution environments. *arXiv preprint 1903.00449*, 2019.

[216] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? In *IEEE S&P*, pages 198–214, 2022.

[217] Charles Rackoff and Daniel R Simon. Non-interactive zero-knowledge proof of knowledge and chosen ciphertext attack. In *CRYPTO*, pages 433–444, 1991.

[218] Geoffrey Ramseyer and Ashish Goel. Fair ordering via streaming social choice theory. In *WINE*, 2024.

[219] Ruben Recabarren and Bogdan Carbunar. Tithonus: A bitcoin based censorship resilient system. *Proc. Priv. Enhancing Technol.*, 2019(1):68–86, 2019.

[220] Michael K. Reiter and Kenneth P. Birman. How to securely replicate services. *ACM Trans. Program. Lang. Syst.*, 16(3):986–1009, 1994.

[221] Team Rocket, Maofan Yin, Kevin Sekniqi, Robert van Renesse, and Emin Gün Sirer. Scalable and probabilistic leaderless bft consensus through metastability, 2019. arXiv:/1906.08936.

[222] Tim Roughgarden. Transaction fee mechanism design. *Journal of the ACM*, 71(4):1–25, 2024.

[223] Walter Rudin et al. *Principles of mathematical analysis*, volume 3. McGraw-hill New York, 1964.

[224] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *FOCS*, pages 543–553, 1999.

[225] Kazue Sako. An auction protocol which hides bids of losers. In *PKC*, pages 422–432, 2000.

[226] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In *CRYPTO*, pages 239–252, 1989.

[227] James Schummer. Manipulation through bribes. *Journal of Economic Theory*, 91(2):180–198, 2000.

[228] Srinath Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *CRYPTO*, pages 704–737, 2020.

[229] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[230] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. Bft protocol forensics. In *CCS*, pages 1722–1743, 2021.

[231] Elaine Shi, Hao Chung, and Ke Wu. What can cryptography do for decentralized mechanism design. *arXiv preprint arXiv:2209.14462*, 2022.

[232] Ron Siegel. All-pay contests. *Econometrica*, 77:71–92, 2009.

[233] Signal website. `https://signal.org`, 2020.

[234] Smith criterion, 2021. `https://en.wikipedia.org/wiki/Smith_criterion`.

[235] Shi-Feng Sun, Man Ho Au, Joseph K Liu, and Tsz Hon Yuen. RingCT 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In *ESORICS*, pages 456–474, 2017.

[236] Weizhao Tang, Lucianna Kiffer, Giulia Fanti, and Ari Juels. Strategic latency reduction in blockchain peer-to-peer networks. *Proc. ACM Meas. Anal. Comput. Syst.*, 7(2):32:1–32:33, 2023.

[237] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM J. Comput.*, 1(2):146–160, 1972.

[238] Michael Bedford Taylor. The evolution of bitcoin hardware. *Computer*, 50(9):58–66, 2017.

[239] Langston Thomas. Fractional NFTs: The good, the bad, and the weird. *NFT Now*, 26 Apr. 2022.

[240] Mohammad Amin Vafadar and Majid Khabbazian. Condorcet Attack Against Fair Transaction Ordering. In *AFT*, pages 15:1–15:21, 2023.

[241] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security*, pages 991–1008, 2018.

[242] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice, 2020. https://sgaxe.com/files/SGAxe.pdf.

[243] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *SRDS*, pages 135–144, 2009.

[244] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient Byzantine fault-tolerance. *IEEE Transactions on Computers*, 62(1):16–30, 2013.

[245] Riad S. Wahby, Ioanna Tzialla, Abhi Shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zksnarks without trusted setup. In *IEEE S&P*, pages 926–943, 2018.

[246] Anton Wahrstätter, Liyi Zhou, Kaihua Qin, Davor Svetinovic, and Arthur Gervais. Time to bribe: Measuring block construction market, 2023.

[247] Joseph Weinberg. NFTs and compliance: Why we need to be having this conversation. *Cointelegraph*, 21 Jan. 2022.

[248] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. A flash(bot) in the pan: measuring maximal extractable value in private pools. In *IMC*, page 458–471, 2022.

[249] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2021.

[250] E Glen Weyl, Puja Ohlhaver, and Vitalik Buterin. Decentralized society: Finding web3's soul. *Available at SSRN 4105763*, 2022.

[251] Wonderproxy. A day in the life of the internet, 2020. https://wonderproxy.com/blog/a-day-in-the-life-of-the-internet/.

[252] XDA Forums. Discussion: Play Integrity API. https://forum.xda-developers.com/t/discussion-play-integrity-api.4479337/, 2022.

[253] Bowen Xue and Sreeram Kannan. Travelers: A scalable fair ordering bft system, 2024.

[254] Sen Yang, Fan Zhang, Ken Huang, Xi Chen, Youwei Yang, and Feng Zhu. Sok: Mev countermeasures: Theory and practice, 2022.

[255] Andrew C Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.

[256] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *PODC*, pages 347–356, 2019.

[257] Mark Zhandry. Optimal traitor tracing from pairings. Cryptology ePrint Archive, Paper 2024/867, 2024.

[258] Jianting Zhang, Wuhui Chen, Sifu Luo, Tiantian Gong, Zicong Hong, and Aniket Kate. Front-running attack in sharded blockchains and fair cross-shard consensus. In *NDSS*, 2024.

[259] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In *OSDI*, pages 633–649, 2020.

[260] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc V Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. In *IEEE S&P*, 2021.