Themelio: a new blockchain paradigm

by

Yuhao Dong

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, Canada, 2023

Examining Committee Membership

The following served on the Examining Committee for this thesis. The decision of the Examining Committee is by majority vote.

External Examiner:	Andreas Veneris Professor, Dept. of E.C.E and C.S., University of Toronto
Supervisor(s):	Raouf Boutaba Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Ian Goldberg Professor, Cheriton School of Computer Science, University of Waterloo
Internal Member:	Sergey Gorbunov Associate Professor, Cheriton School of Computer Science, University of Waterloo
Internal-External Member:	Wojciech Golab Associate Professor, Dept. of E.C.E., University of Waterloo

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

This dissertation includes first-authored, peer-reviewed materials that have appeared in various conference proceedings.

The following list serves as a declaration of the Versions of Record for works included in this dissertation:

- *Portions of Chapter 5*: **Dong, Yuhao**, and Boutaba, R. "Melmint: trustless stable cryptocurrency" in *Cryptoeconomic Systems*, vol 1, no 1.
- *Portions of Chapter 5*: **Dong, Yuhao**, and Boutaba, R. "Elasticoin: Low-volatility cryptocurrency with proofs of sequential work." In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)* (pp. 205-209). IEEE.
- *Portions of Chapter 6*: **Dong, Yuhao**, Goldberg, I., Gorbunov, S., and Boutaba, R. "Astrape: Anonymous Payment Channels with Boring Cryptography." In *Applied Cryptography and Network Security: 20th International Conference, ACNS 2022, Rome, Italy*, June 20–23, 2022, Proceedings (pp. 748-768). Cham: Springer International Publishing.
- *Portions of Chapter 7*: **Dong, Yuhao**, Kim, W., and Boutaba, R. (2018, November). "Bitforest: a portable and efficient blockchain-based naming system." In *2018 14th International Conference on Network and Service Management (CNSM)* (pp. 226-232). IEEE.

Abstract

Public blockchains hold great promise in building protocols that uphold security properties like transparency and consistency based on internal, incentivized cryptoeconomic mechanisms rather than preexisting trust in participants. Yet user-facing blockchain applications beyond "internal" immediate derivatives of blockchain incentive models, like cryptocurrency and decentralized finance, have not achieved widespread development or adoption.

We propose that this is not primarily due to "engineering" problems in aspects such as scaling, but due to an overall lack of *transferable endogenous trust*—the twofold ability to uphold strong, internally-generated security guarantees and to translate them into application-level security. Yet we argue that blockchains, due to their foundation on game-theoretic incentive models rather than trusted authorities, are uniquely suited for building transferable endogenous trust, despite their current deficiencies. We then engage in a survey of existing public blockchains and the difficulties and crises that they have faced, noting that in almost every case, problems such as governance disputes and ecosystem inflexibility stem from a lack of transferable endogenous trust.

Next, we introduce Themelio, a decentralized, public blockchain designed to support a new blockchain paradigm focused on transferable endogenous trust. Here, the blockchain is used as a low-level, stable, and simple root of trust, capable of sharing this trust with applications through scalable light clients. This contrasts with current blockchains, which are either applications or application execution platforms. We present evidence that this new paradigm is crucial to achieving flexible deployment of blockchain-based trust.

We then describe the Themelio blockchain in a detail, focusing on three areas key to its overall theme of transferable, strong endogenous trust: a traditional yet enhanced UTXO model with features that allow powerful programmability and light-client composability, a novel proof-of-stake system with unique cryptoeconomic guarantees against collusion, and Themelio's unique cryptocurrency "mel", which achieves stablecoin-like low volatility without sacrificing decentralization and security.

Finally, we explore the wide variety of novel, partly off-chain applications enabled by Themelio's decoupled blockchain paradigm. This includes Astrape, a privacy-protecting off-chain micropayment network, Bitforest, a blockchain-based PKI that combines blockchain-backed security guarantees with the performance and administration benefits of traditional systems, as well as sketches of further applications.

Acknowledgements

I am greatly thankful for my supervisor Prof. Raouf Boutaba for his encouragement and support. When I was a new PhD student, he did not hesitate to help me learn the ropes of academic research whenever I needed him, and throughout the later years he was always available for brainstorming ideas and discussing research. Prof. Boutaba always gave me the freedom to pursue my ideas and supported me fully, and was an excellent mentor in navigating the initially bewildering world of CS academia and research collaborator. I am also very grateful to Prof. Noura Limam for convincing me to pursue a PhD at Waterloo, a decision I am very happy I took.

I also thank Woojung Kim, my office-mate for the first two years of my PhD, with whom I have had many very fruitful discussions and research collaborations. In fact, our discussions on PKIs and roots of trust was what eventually led me down the rabbit hole and realize the yet unrealized, paradigm-shift potential of public blockchains. I also greatly enjoyed my discussions with other members of the Networks Lab, such as Milad, Shihab, Reaz, and Haibo, on subjects ranging from cryptographic protocols to the minutiae of Asian internet routing patterns.

I especially thank Prof. Ian Goldberg for his help and mentorship during our research into payment channels. He really went above and beyond in helping me with the Astrape paper, not just with straightforward yet very constructive criticism, but often directly lending a hand in writing key passages, especially when math was involved! A significant portion of this dissertation would not likely be possible without his help. I also thank him, as well as Profs. Sergey Gorbunov and Wojciech Golab for taking the time to participate in the committee and reviewing this dissertation.

In addition, I'd like to thank the Polychain team for giving me the opportunity to develop and use my research in production, which proved invaluable to crystallizing the ideas found in my dissertation. In particular, I am greatly thankful to Niraj Pant for somehow finding a YouTube video of me presenting at a conference compelling enough to invite me to an accelerator, as well as Will Wolf for mentoring me through the process of translating a loose collection of research ideas into a production system.

Finally, I am the most indebted to my parents. They cultivated in me from a young age a love for learning, and they did everything they could to give me the education that would best shape my life for the better. I am especially thankful for my mother's continual and exceptional spiritual mentorship. Without her wisdom and guidance, my inchoate childhood curiosity could not possibly have developed into an appreciation of learning and rational inquiry as transformative, productive, and even divine — an understanding that was indispensible for guiding my life, both academic and non-academic, through many difficulties and doubts throughout the years.

Dedication

AMDG. To Thisbe.

Table of Contents

List of Figures

List of Tables

Chapter 1

Why another blockchain?

1.1 A "blockchain revolution"?

The promise of blockchains

Trust on the Internet is a rare commodity. Participants are often anonymous, communication is inherently insecure and everyone is at most a few hundred milliseconds away from potential attackers. Like any other setting where trust is absent, the Internet forces us to either only transact with parties we already trust offline whose digital identities can be cryptographically verified, or bootstrap security through a preexisting *root of trust* — an entity or mechanism that is universally trusted.

As the former is very difficult to find without costly offline ceremonies like key-signing parties, security on the Internet nearly universally depends on the latter. At present, the most common root of trust on the Internet is some sort of centralized institution. For instance, Transport Layer Security (TLS), the protocol that underpins the vast majority of encrypted network traffic, relies on root certificate authorities (CAs) that issue certificates that attest to the public keys belonging to hostnames.

Unfortunately, these central points of trust are often central points of failure. Compromised root CAs lead to catastrophic meltdowns of the basic cryptography of the encrypted Web [PC11]. Poisoned DNS servers enable defacing high-profile websites. Hacked update servers can instantly distribute malware to enormous numbers of unsuspecting computers, crippling critical systems [RN17].

Moreover, centralized control of the "commanding heights" of our modern interconnected

society lends a disproportionate amount of power to a small oligarchy of trusted service providers. This enables a wide range of abuse with devastating real-world consequences. Centralized social media platforms can insidiously manipulate and censor user communication [Sun18]. Governments can build Orwellian surveillance systems [Wan17] by aggregating massive amounts of data from centralized sources.

In the face of these numerous perils of trust centralization on the Internet, blockchains offer an attractive alternative. Public blockchains like Bitcoin [Nak08] and Ethereum [Woo14] are designed to be unforgeable, append-only ledgers accessible to all. They provide secure, transparent, and permanent records of transactions while being completely decentralized. Instead of depending on centralized services, applications such as public key infrastructures, document timestamping services, and electronic money can now use this shared ledger to guarantee security.

This has powered a large growth of interest and optimism in blockchain technology. Bestselling books like *Blockchain Revolution* tout utopian dreams of a secure and transparent Internet backed by blockchain trust. Expectations of massive growth in blockchain usage have sparked a boom in blockchain-based cryptoassets of mind-boggling proportions — the total market value of cryptocurrencies grew from around $3 billion USD in 2015 to $500 billion by 2018 and an astonishing $1 trillion by 2023.

The curious absence of blockchain applications

Yet despite all the hype of a blockchain revolution, production systems using public blockchains remain rather rare. Most popular blockchain use cases are in some way or another *about blockchains themselves*: cryptocurrency DeFi (decentralized finance), trading apps, blockchain viewers, etc.

On the surface, this lack of usage comes from "engineering" deficiencies in current blockchain technology. For example, users who wish to achieve full security must synchronize and store the blockchain's entire transaction history (which may be dozens of terabytes or more), while Nakamoto consensus as used in some blockchains like Bitcoin takes hours for transactions to become securely irreversible. In addition, fluctuating cryptocurrency prices create currency risk, while congestion leads to spikes in transaction fees. Few of these shortcomings are acceptable in modern production systems.

Furthermore, public blockchains by nature require unanimous agreement on the blockchain protocol. Therefore, protocol upgrades are often contentious and sometimes disruptive to software ecosystems, threatening application-level stability.

Finally, mainstream blockchains often lack a way for non-blockchain code to verifiably access blockchain data without either synchronizing the entire blockchain history or trusting a third party

(in more technical terms, they lack *trustless light clients*). Querying blockchain data in the current ecosystem thus typically involves asking some third-party provider, such as Infura [Inf23]. This can pose serious trust issues, such as centralized, unaccountable censorship [Coi22] that blockchains were intended to avoid.

Attempts to fix these technical issues, unfortunately, often further weaken the promise of trustless, protocol-backed trust. One particularly obvious example is the idea of a "private blockchain". Private blockchains, like those based on Hyperledger Fabric [Cac16], do use append-only distributed ledgers similar to those of public blockchains, but they are generally deployed within an environment isolated from public access, such as across a corporate WAN or even inside a single datacenter.

Enterprise applications, such as supply-chain tracking or processing business payments, have an especially strong tendency towards using private blockchains. This is because unlike their public counterparts, private blockchains promise performance and reliability comparable with traditional databases. For this reason, private blockchains have been adopted by a wide variety of production systems, ranging from the SecureKey identity service [Sec] to Estonian government systems [E-E].

However, private blockchains give up most of the security, transparency, and decentralization of public blockchains. *Consortium* blockchains run by a few mutually distrusted parties are an attempt at a compromise, but absent the sort of robust incentive mechanisms that uphold public blockchains, it is difficult to meaningfully separate their security properties from that of a private blockchain run by a corporate consortium. In summary, since private blockchains sacrifice decentralization in pursuit of usability, they are unlikely to be key to ushering in a more decentralized Internet.

Other attempts at attacking the technical barriers to blockchain usage encounter similar issues. Increasing blockchain throughput, for instance, often leads to designs like EOS's delegated proof-of-stake [Flo18] that greatly weaken security, or designs like Solana [Sol23] that further weaken light-client support and entrench centralized API providers compared to Ethereum or Bitcoin. Value-stable cryptocurrencies ("stablecoins"), which try to fix the issues that cryptocurrency volatility poses to a decentralized economy, end up requiring centralized, "oracle"-based pegs to external fiat currencies like the US dollar.

1.2 What's wrong with blockchains?

A "blockchain trilemma"

A prevalent way of framing these difficulties in blockchain design in the literature is by positing a *blockchain trilemma*. Paraphrasing its formulation by Ethereum founder Vitalik Buterin [But21], it is the claim that blockchains cannot easily possess all three of:

- *Scalability*: the ability to process more transactions than a single consumer-grade computer can verify
- *Decentralization*: lack of any trust dependencies on a small group of large centralized actors
- *Security*: ability to resist attacks that compromise a large fraction (>25%) of participating nodes

In this framework, private blockchains have scalability and security,[1] but completely lack decentralization. First-generation blockchains like Bitcoin pick decentralization and security, yet do not achieve scalability. If left unsolved, the blockchain trilemma poses a severe, if not fatal, obstacle to widespread blockchain adoption. Global adoption seems to require high scalability, yet the trilemma puts that in opposition to decentralization and security, both of which are critical for the revolution in trust that blockchains promise.

Much recent research on improving blockchains focuses on attempts to solve the blockchain trilemma. For example, a solution often considered to be the most promising is blockchain "sharding". In a sharded blockchain, instead of a global broadcast bus of transactions, the blockchain is split into many subchains, with intricate cryptoeconomic mechanisms preventing attackers from compromising the security of even one subchain without defeating a global majority. This achieves high scalability and decentralization while only slightly weakening security guarantees compared to traditional blockchains, at the cost of significantly increased protocol complexity.

A more serious deficiency: transferable endogenous trust

But we think that the blockchain trilemma, though important for understanding the difficulties of scaling up blockchain throughput, glosses over a much more serious problem with current blockchains — their *lack of endogenous trust*. Endogenous trust refers to the fact that the security of a public blockchain does not rely on preexisting trust in the honesty[2] of any participant. Instead, trust is backed by cryptoeconomic mechanisms that we can show incentivizes specified behavior given only general assumptions about human action, like the fact that all other things being equal, humans prefer monetary rewards to punishments.

Endogenous trust is the single most precious property of blockchains that enables secure applications with properties elusive to pre-blockchain systems. For example, someone using a bank must trust the bank no matter what form of communication protocol is used, yet a Bitcoin user does not need to even know which miner ends up processing their transaction, let alone establish some

[1] Note that here, the "participating nodes" would be fault-tolerant replicas deployed by trusted parties.

[2] Here, we use "honesty" in the usual technical sense of "behaves in the protocol-specified manner".

sort of trust with that miner. Instead, the cryptoeconomics of Bitcoin mining internally incentivize miners to cooperate in such a way that makes the Bitcoin protocol as a whole trustworthy.

Unfortunately, we very often see disruptive failures in endogenous trust. Contentious governance problems like the Bitcoin block-size controversy of 2017 or the "DAO fork" that split Ethereum into Ethereum and Ethereum Classic arise when external factors incentivize users to override a blockchain's endogenous trust mechanism. Poor internal incentives cause out-of-band coordination to become necessary to prevent game-theoretical issues like "SPV mining" in Bitcoin. Even volatile cryptocurrency prices can be analyzed as a lack of an endogenously trustworthy store of value, forcing users to reach for fiat-pegged stablecoins like Tether that altogether forgo endogenous trust.

In Section 2.1, we investigate this issue and find two important problems. Lack of endogenous trust in first-generation blockchains generally comes from *application-blockchain friction*, where blockchains are tightly coupled to applications and inherit much of the instability that comes with application-level changes and failures. Such instability then endangers endogenous trust, as an unstable protocol requires constant external governance that must rely on exogenous trust. For example, the "DAO fork" is a response to the failure of The DAO, an Ethereum application, and the great majority of "routine" consensus-breaking upgrades to blockchains are for adding features needed by newer applications.

In addition, the limited endogenous trust produced by current blockchains is further hampered by *poor trust transference* — they may have strong endogenous trust, but cannot easily use this trust to bootstrap application-level trust. They lack what we call *transferable endogenous trust*, a term we will discuss subsequently.

In summary, we see that a poor blockchain abstraction prevents transferable endogenous trust: current blockchains present models that are at once too "convenient" for current applications, yet too limited to fully confer their security properties to applications. Evidently, a new approach to blockchain design is desperately needed.

1.3 Towards a solution

We propose Themelio, a new blockchain focused on achieving robustly transferable endogenous trust through *neutrality-maximizing abstraction*. This refers to a "Goldilocks zone" of blockchain abstraction, where the protocol is both featureful enough to allow for a wide range of applications, while low-level enough to be decoupled from the needs of, and neutral between, *specific* applications. This allows for extremely long-term protocol stability and immutability — key for blockchain security — without compromising on flexibility. We argue that the Internet Protocol, which stayed stable for decades despite dramatic changes in both applications and underlying telecom technology,

is an analogous example of neutrality-maximizing abstraction. Even though no current blockchain exists in this zone, we show that a new blockchain exploiting this effect can be constructed, through key innovations in data model, consensus, and cryptocurrency issuance.

The structure of this thesis is as follows. In Chapter 2, we present a survey of the existing literature on public blockchains. Through this survey, we first argue that transferable endogenous trust is the key innovation of public blockchains implicit in analyses of blockchain security, even though it has never to our knowledge been exactly formulated in the existing literature. We demonstrate that other commonly cited features, like "decentralization" and "immutability", do not capture the novel security properties of public blockchains. We then take a look at the development of blockchain technology up to this point through the lens of transferable endogenous trust. We see that despite steady technological advancement in attacking traditional hard problems like the "blockchain trilemma", endogenous trust remains weak.

Chapter 3 introduces Themelio's overall *state model*: the basic abstraction on which blockchain transactions operate. We argue that a bare-bones "coin-based" transaction model, coupled with a few innovations that enable powerful trust transference, produces a neutrality-maximizing blockchain design. In Chapter 4, we turn our attention to Synkletos, Themelio's *consensus game* — the consensus protocol and its supporting cryptoeconomic incentives. Synkletos moves away from traditional forms of consensus game design, which tend to model blockchain consensus as a fault-tolerance problem with limits on participant collusion. Instead, Synkletos facilitates collusion between consensus participants in setting protocol parameters, while constraining these parameters such that collusive, monopolizing behavior is beneficial. This gives us a robustly collusion-tolerant consensus game, which we argue is critical for long-term consensus safety in endogenous trust protocols. Chapter 5 describes Melmint, the issuance mechanism for Themelio's base currency Mel. Melmint provides the first known *trustless stablecoin*: a cryptoasset that keeps a relatively stable purchasing power, yet relies solely on endogenous trust. This achievement derives from replacing the oracles pervasive in stablecoin design with a novel and trustlessly measurable index of sequential computation cost, the DOSC (day of sequential computation). We show through both heuristic arguments and simulations that Mel can keep a stable value in a wide variety of market conditions, and argue that a trustless stablecoin is critical not only for daily use by cryptocurrency users, but also for designing truly autonomous, endogenous-trust financial systems.

Finally, Chapters 6, 7, and 8 dive into case studies of practical applications. We demonstrate how Themelio as an endogenous root of trust can support a wide variety of decentralized applications, many of which cannot be supported by current-generation public blockchains.

Chapter 2

A survey of endogenous trust

In this chapter, we take a look at existing blockchain technology through the lens of endogenous trust — security rooted in cryptoeconomic incentives rather than preexisting human trust, a property that we demonstrate is the key objective of public blockchains. We start by a survey of the extant literature on blockchains, showing that existing surveys and reviews, though pointing out a multitude of open research problems in the field, display a surprising lack of consensus on the actual purpose of blockchains and the standard by which they should be evaluated. Yet through examining the design goals of individual technologies, we can reconstruct endogenous trust as the central *implicit* goal of almost every blockchain system. We formulate a tentative definition of endogenous trust, as well as *transferable* endogenous trust, a stronger property that requires that the incentive-based security of the blockchain can be used to bootstrap security for other systems.

Armed with this goal, we examine the history of blockchain technological development, both "classical" blockchains like Bitcoin and Ethereum 1.0 and new designs like Ethereum 2.0, Tezos, and Celestia based on unconventional approaches to scaling, governance, and other key features. We see that classical blockchains, following either a "application is the blockchain" model or an "application on the blockchain" model, suffer from tight coupling between applications and blockchains. This leads to two undesirable features — complexity and mutability — that preclude strong endogenous trust. We then see that most newer blockchains, though solving many important technical problems, do not directly address this tight coupling and in fact often worsen it. Nevertheless, two later developments do attempt to address blockchains' key goal: on-chain governance and minimal "data-availability" protocols. Unfortunately, we see that the first fails to capture the essence of endogenous trust, and the latter, though amenable to much stronger endogenous trust than classical blockchains, does so by crippling transferable endogenous trust and making it difficult to build blockchain applications with the strong security of the blockchain itself.

Finally, we lay out some concrete takeaways of this survey that will then serve as a guide to developing a novel blockchain that is truly optimized for transferable endogenous trust.

2.1 Endogenous trust: the implicit goal

Fundamental blockchain goals? A sad "meta-SoK"

A possible route for uncovering the most important goals of blockchains is to survey existing systematizations of knowledge (SoKs) on blockchain technology. This is because formal design goals tend to take center stage in surveys of a field. No SoK of secure communication, for instance, is complete without a discussion of the "CIA" (confidentiality, integrity, authenticity) triad.

Table 2.1: A selection of blockchain surveys

Work	Citations	Notes
Zheng et al., 2018 [Zhe+18]	1914	Key features: *decentralization*, *persistency*, *anonymity*, *auditability*. Taxonomy: *consensus determination*, *read permission*, *immutability*, *efficiency*, *centralized*, *consensus process*
Yli-Huumo et al., 2016 [Yli+16]	1587	Does not identify features other than *"distributed database solution that maintains a continuously growing list of data records that are confirmed by the nodes participating in it."*. Identifies some hard problems like throughput, latency, usability, etc.
Li et al., 2020 [Li+20]	936	Key features: *irreversible and traceable*, *decentralized and anonymous*, *secure and permissionless*, *fast and global*. Attempts a general taxonomy of all blockchain security threats.
Lin and Liao, 2017 [LL17]	851	Key features: *decentralized*, *transparent*, *open source*, *autonomous*, *immutable*, *anonymous*.
Wüst and Gervais, 2018 [WG18]	789	Key features: *public verifiability*, *transparency*, *privacy*, *integrity*, *redundancy*, *trust anchor*. Proposes systematic flowchart to decide "do you need a blockchain".
Yaga, et al., 2018 [Yag+18]	522	Key features: *ledger*, *secure*, *shared*, *distributed*. Comprehensive NIST review of blockchain tech.

Unfortunately, within the SoK literature it turns out there is a surprising lack of consensus, or even discussion, on the core goals of blockchain systems, even though there are many individual technologies developed to fix individual problems. This can be seen through an overview of the most highly cited surveys and reviews of the field, summarized in Table 2.1. Generally, we see loosely organized lists of the research challenges that current work attacks, accompanied by taxonomies of blockchains based on particular features. The closest thing to descriptions of blockchains' core

goals we see is usually a collection of "good" blockchain characteristics, like decentralization and auditability, that often do not coincide with similar lists given by other surveys.

For example, one of the highest-cited blockchain SoKs is "Blockchain Challenges and Opportunities: A Survey" by Zheng et al. [Zhe+18]. It simply identifies some features that some blockchains have: decentralization, persistence, anonymity, and auditability, then proceeds to build a rather unsystematic taxonomy of blockchains on axes like immutability and centralization. Yet Lin and Liao [LL17] do not treat autonomy and immutability as key blockchain goals — autonomy does not appear in the work of Zheng et al., while immutability is treated as a taxonomic category that blockchains may or may not have. Even the highly comprehensive NIST review of Yaga et al. [Yag+18] only managed to give a precise definition of a blockchain — a "secure", "shared", and "distributed" "ledger" — without much discussion of *why* anyone would need a secure, shared, distributed ledger, or what properties must such a ledger have to be useful.

Worse, even these descriptions of individual features are often unhelpful or inaccurate whenever they touch on the purpose behind any given feature. For example, Zheng et al. claim that through decentralization, "blockchain[s] can significantly reduce the server costs and mitigate the performance bottlenecks at the central server" [Zhe+18]. This assertion is incorrect — unlike systems without consistency requirements like BitTorrent, strongly consistent, fully replicated decentralized systems like blockchains have *worse* performance than centralized systems — as well as missing the crucial security and fault-tolerance purposes of decentralization. Unfortunately, this inaccurate claim about blockchain decentralization can be found in other surveys too, such as Monrat et al. [MSA19].

For whatever reason, existing academic SoKs do not seem to come to a strong consensus on the most important features of blockchain design. How can we make any claim about what features are key to blockchains then?

Reconstructing transferable endogenous trust

One possible conclusion is that there is not much to blockchains, and the widespread interest in their use is merely unfounded hype. Instead, we propose that there does exist a unifying aim behind all of the disparate blockchain features, and it is implicit in both the earliest discussions of blockchains, as well as later discussions of *attacks* upon blockchains.

In the original Bitcoin paper [Nak08] by the pseudonymous "Satoshi Nakamoto", the concept of a blockchain (originally called a "peer-to-peer distributed timestamp server") was first introduced. Nakamoto expressed frustration at the pervasive need for preexisting trust in traditional financial transactions and described Bitcoin's blockchain as "an electronic payment system based on cryptographic proof instead of trust".

In other words, Bitcoin was designed to behave securely as long as its cryptography and incentive design was secure, *independent of trust in those who run the protocol*. This is a design goal that is difficult to find in pre-blockchain computer systems. In particular, other protocols' security properties typically involve protecting certain honest users, assumed to follow the protocol, against adversaries with particular powers which may control some particular other parties. For example, a secure communications protocol allows Alice to mail an encrypted letter to Bob without trusting the mailman Charlie. Yet Alice must trust that Bob will also follow the protocol, and not go on to gossip about the letter to Charlie! Here, preexisting trust between Alice and Bob is thus *transferred* into trust in the communication system relayed through Charlie, despite Charlie being untrusted, but the protocol cannot bring Alice and Bob to trust each other in the first place.

This avoidance of trust in participants is also implicit in the security models of the literature discussing attacks on blockchain security. These attacks almost always focus on attacks that refute a blockchain's claim to be "based on cryptographic proof instead of trust". For example, the famous 2016 "selfish mining" paper by Sapirshtein et al. [SSZ16] used a security model that assumes conditions like miners not coordinating their actions and being self-interested, but notably, it did not trust that miners or even some subset of miners will actually follow the protocol. It then goes on to present attacks that can break Bitcoin's consensus safety even under these conditions. This again stands in contrast to other protocols: we usually do not consider attacks on TLS without first assuming that the participants whose communications we wish to protect do actually run TLS.

We name this property of creating security "within" the protocol *endogenous trust*:

> Definition 1: A system has *endogenous trust* if the assumptions it makes about participant actions to achieve its desired behavior come with protocol-internal economic arguments for why rational participants will take those actions.

An important note is that endogenous trust is not a property of a security model per se, but describes how it is *justified*. For instance, consensus algorithms used in cryptocurrency blockchains follow a variety of threat models — Bitcoin requires that less than 1/2 of hashpower is adversarial [Nak08], while Algorand assumes an attacker which can adaptively corrupt the owners of less than 1/3 of staked assets [Gil+17]. But all of these protocols contain incentives that back a *protocol-internal argument* for why these properties will hold.

In this sense, endogenous trust ultimately avoids making a distinction between honest users whose use of the system must be protected against adversaries who attempt to attack it, but treats every participant at the beginning as an economically motivated potential adversary. Only after an analysis of incentives do we then arrive at a situation where certain participants might be assumed

honest. For instance, Bitcoin mining is usually analyzed under a model where *every single miner* is assumed to use whatever strategy maximizes profit, whether or not that strategy honestly follows the Bitcoin protocol, but then we generally want to show that less than 1/2 of hashpower will choose an adversarial strategy.

We generally construct endognous trust using cryptoeconomic mechanisms like mining incentives, where cryptographic verification is coupled with economic mechanism design to incentivize participants towards "honest" behavior. Combined with general assumptions of rational choice when confronted with differing incentives, cryptoeconomics allows us to conclude that participants will uphold a particular system's security goals without any prior assumptions of honesty.

A more narrowly defined property, yet a property crucial to the actual usefulness of endogenous trust, we call *transferable endogenous trust*:

> Definition 2: A system with endogenous trust has *transferable endogenous trust* if the security properties of a wide range of user-facing applications can be reduced to the security of the system.

Of course, this definition is necessarily less precise due to the ill-defined nature of a "wide range" of applications. But transferable endogenous trust captures an important notion: a blockchain, or any other kind of protocol, is not very useful if it does not secure actual user-facing applications. For example, if Bitcoin could not support secure money transfers, but can only produce useless, empty blocks, it would lack transferable endogenous trust even if it has very strong endogenous trust and participants are robustly incentivized to correctly produce the empty blocks.

Transferable endogenous trust as the central goal of blockchains greatly illuminates much of the confusing literature on blockchain research challenges. The blockchain trilemma of scalability/decentralization/security, for example, is much better understood as *endogenous trust is hard to transfer at scale*. "Decentralization" plus "security" almost always means transferable endogenous trust rather than merely having a large number of participants and being resistant to a significant fraction of dishonest participants. A blockchain where all blockchain histories are considered valid will technically be "decentralized" and "secure", but would not be considered a viable blockchain due to a lack of any endogenous trust transferable to applications. Identifying decentralization and security as the unitary concept of transferable endogenous trust also explains why the usual examples for the "scalable+secure" and "scalable+decentralized" edges of the triangle are not blockchains in the proper sense at all — private and consortium blockchains for the former, multi-blockchain ecosystems for the latter — they lack the core feature of a useful public blockchain.

Another important source of confusion that we can clarify is the common characterization of blockchains as "trustless" or "autonomous". Advocates of a blockchain-powered fundamental reorganization of Internet trust, like the authors of the bestselling *Blockchain Revolution* [TT16], often talk of blockchains as if they were infallible rule-enforcing automata that can finally rid the world of trusting humans. At this point, cooler heads point out that blockchain users still trust many humans — whoever built their computers, wrote their operating system, etc. — and often discount the idea of blockchains eliminating trust [Yag+18] or at best wave to some weak notion of trust "minimization".

Yet endogenous trust shows us that the "trustless" claim is not without merit. Blockchain trust, even where they rely on trusting third parties such as miners, is fundamentally based on blanket *rationality* assumptions rather than assumptions about honest vs. adversarial users. We do see a qualitative difference in trust that promises changes in security modeling across diverse applications.

2.2 Classical blockchains

Now that we have distilled the core design goal of blockchains, we can move on to an overview of blockchain development from the standpoint of transferable endogenous trust. We start our discussion with *classical blockchains* — blockchains that share the same high-level architecture as the first blockchain Bitcoin. In particular, we look at systems with transactions collated into a linear series of globally broadcast blocks, as well as permissionless, probabilistic, heaviest-chain-rule "Nakamoto consensus". We see that although they do achieve some measure of endogenous trust, classical blockchains significantly fall short of fully transferable endogenous trust.

Bitcoin: endogenous trust for money

As the first-ever blockchain, Bitcoin [Nak08] is essentially an implementation of *endogenously trustworthy money*. Nakamoto wished to create a peer-to-peer payment network whose correct functioning does not depend on trust in participants. Towards this end, Bitcoin uses a cryptoeconomic mechanism to incentivize "miners" to collaborate in building a coherent block history. This probabilistic *Nakamoto consensus* boils down to a few extremely simple principles:

- The first block, the *genesis block*, is hardcoded in the protocol definition.
- Anybody can at any time broadcast a block. Each block must refer to a *parent* block by the parent's cryptographic hash. This forms a *block tree* consisting of all blocks ever broadcast.

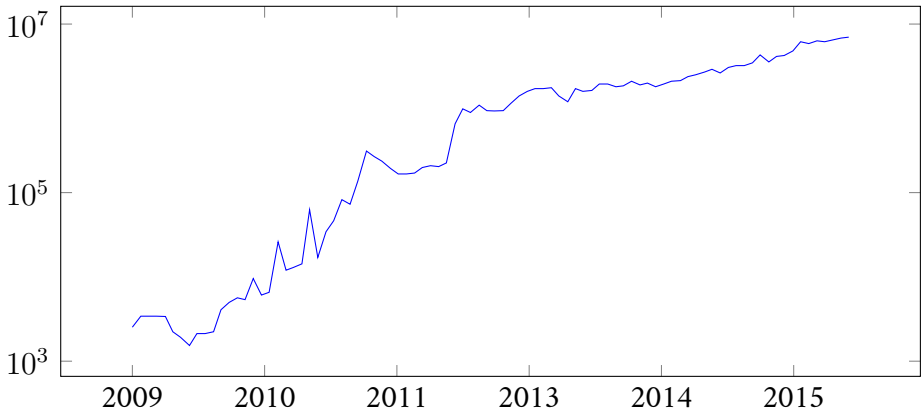


Figure 2.1: Early growth in monthly Bitcoin transaction volume

- Each block contains a *proof of work* (PoW) that demonstrates that the block creator — the miner — "wasted" a certain amount of computation, as well as a *block reward* that rewards the miner with a protocol-defined sum of newly created cryptocurrency.
- The canonical *blockchain* is defined as the branch of the block tree, consisting entirely of valid blocks, with the largest total amount of work (the "heaviest" chain).

The intuition behind Nakamoto consensus is that miners attempt to produce blocks that will be part of the canonical blockchain, as this is the only way that their block rewards will be honored by Bitcoin users. Thus, miners are incentivized to strive to produce blocks that extend the current canonical blockchain, since the current canonical blockchain is also the most likely branch to be "heaviest" in the future. This then results in a linear blockchain, with minor "orphaned" branches due to network latency, rather than a limitlessly growing block tree, all powered by endogenous trust.

After its invention, Bitcoin quickly became a widely traded financial asset. By 2013, a mere 5 years after its inception, the total value of all bitcoins reached $9 billion USD with a daily trading volume of $20 million USD [Coi]. On-chain transaction volume also grew exponentially, reaching 1 million transactions per month by 2013, as shown in Figure 2.1. Merchants needing irreversible transactions, investors seeking an asset unaffected by central bank policy, and (more controversially) participants in darknet markets like Silk Road and people seeking to evade capital controls or taxes all found Bitcoin's endogenous trust indispensable — and seemingly unassailable.

The first cracks appear Unfortunately, it did not take very long for weaknesses in Bitcoin's endogenous trust to appear. Very early on, it was noted that the proof-of-work consensus (or

"mining") used in Bitcoin provides an extremely uneven income stream to miners due to its inherently probabilistic nature. Starting in 2010 with Slush Pool[1], miners began to form *mining pools* where they pool their resources to act as a single miner and split the rewards according to the amount of work contributed. Since the managers of a mining pool collectively decide the use of its resources, the proliferation of mining pools drastically reduces the effective number of consensus participants. This increase in centralization makes out-of-band trust in mining pools — rather than endogenous trust in Bitcoin's mining protocol — increasingly important.

By July 2014, mining centralization progressed to such an extent that a single mining pool, GHash.IO, controlled more than 51% of Bitcoin's compute power. This undermined Bitcoin's security model, making it dependent on trusting a single mining pool. A flurry of activity in the community ensued [Haj21]: the price of a bitcoin dropped by 5%. Bitcoin developer Peter Todd sold off half of his holdings. The incident received broad coverage in mainstream media despite Bitcoin being a rather niche application at the time. Catastrophic loss of trust was only averted when GHash.IO committed to voluntarily reduce its hashrate to 40% of global hashrate and asked other mining pools to follow their example. This clearly demonstrates that Bitcoin's endogenous trust mechanisms are insufficient.

Centralization proved not to be the only threat to Bitcoin's endogenous trust. Careful game-theoretical analysis over the years revealed multiple unexpected strategies that allow miners to act in ways that are harmful to Bitcoin security yet profitable to themselves. For instance, *Selfish mining* [ES14] allows miners who control a significant (but less than majority) portion of the hashrate gain higher payoffs by selectively postponing block publication. *SPV mining* involves miners "free riding" on others' block validation, blindly producing blocks without validating the previous block.

Widespread deployment of any of these strategies would greatly disrupt Bitcoin's security and usability. Out-of-band coordination between miners (ironically made possible by the extensive centralization of mining pools) is the only thing keeping adversarial mining strategies from proliferating. In fact, in July 2015 the majority of Bitcoin hashpower briefly mined on an invalid fork due to widespread SPV mining before most miners realized what was happening and switched back to "correct" mining.

At this point, it is obvious that unexpected incentives in Bitcoin's *technical* cryptoeconomic design pushed Bitcoin towards centralization and adversarial mining, gravely wounding its endogenous trust. *Social* problems, however, are even more damaging to Bitcoin's trust.

2017: Bitcoin's "general crisis" As Bitcoin's popularity increased, users began to run into technical limits inherent in its design. Foremost among them was the hardcoded block size limit, which

[1] `https://slushpool.com`

capped the size of each block to 1 MB. This corresponds to no more than approximately 3 transactions per second. By early 2017, almost every block was full, and transaction fees began to rise dramatically. Users were paying up to several dollars per transaction on a system originally designed to reduce transaction costs.

Most people agreed that something has to be done to the protocol — the original Bitcoin protocol obviously cannot scale any further. Here, Bitcoin's failure at endogenous trust is obvious in the community's consideration of an externally enforced, consensus-breaking protocol change. More alarmingly, the community was sharply divided as to what direction the protocol should change towards. Half of the community, including the majority of miners, argued for simply adjusting the block limit upwards. The other half argued that increasing the block limit risks further centralizing blockchain participation. Instead, they supported developing protocols (such as the Lightning Network) that process transactions securely without congesting the blockchain itself. The developers of Bitcoin Core, the most popular implementation of the protocol, stood in the second camp.

At this point, Bitcoin's endogenous trust has pretty much completely broken down. Instead of trusting a self-correcting cryptoeconomic protocol, users of Bitcoin were forced to pick sides in an out-of-band struggle eerily reminiscent of traditional socio-political conflicts that blockchains were supposed to avoid. Many at that time feared that the "block size debate" would entirely discredit the idea of a neutral and trusted blockchain cryptocurrency. [Mor17]

Eventually, the conflict was resolved in an unsatisfying and disruptive way. On August 1st, 2019, the Bitcoin blockchain permanently split into two competing forks — "Bitcoin", which kept smaller blocks but introduced protocol changes to better support off-chain payments, and "Bitcoin Cash", which increased the block limit to 8 MB. This split sent ripples throughout the Bitcoin community as exchanges, wallets, and merchants were suddenly faced with two cryptocurrencies that have different rules but share the same pre-fork history.

Bitcoin's block size fiasco demonstrates an important point: in "application blockchains", tension between blockchain design and evolving application requirements often results in demands for external intervention in the protocol that damage endogenous trust. This is because it is almost impossible to anticipate all future changes in application usage (such Bitcoin's astonishing growth in transaction volume). Therefore, unlike trust failures from cryptoeconomic oversights, trust failures due to "application-blockchain friction" appear inevitable for blockchains designed for a specific application, such as Bitcoin.

Bitcoin's failures to transfer trust Furthermore, even if we grant that Bitcoin provides sufficient endogenous trust, it does not provide what's needed to transfer that to application-level security beyond Bitcoin's native application of sending bitcoins. For example, the decentralized naming

system Blockstack [Ali+16] was unable to fully root their security properties on Bitcoin, in a large part due to the weak feature set of Bitcoin light clients. More complex decentralized systems built on Bitcoin, like the Bisq [Bis] over-the-counter exchange, rely much more extensively on mechanisms (like human arbitration and off-chain order books) that lack Bitcoin's endogenous trust.

Ethereum: endogenously trusted computing

When Ethereum was first released in 2015, it aimed to be a blockchain radically more flexible than Bitcoin, but operating on the same basic principle of Nakamoto consensus. Instead of implementing a single application, Ethereum aims to be the equivalent of an operating system for endogenously trustworthy apps, or "dApps" (decentralized apps). This functionality is powered by a Turing-complete language (EVM) for blockchain-embedded programs, or "smart contracts", that can access and store arbitrary state within the blockchain. Ethereum is effectively a "world computer" that anyone can trustlessly use, with decentralized consensus on the state of this "computer".

Over the subsequent years, a wide variety of Ethereum apps developed, including novel cryptocurrencies like Dai, decentralized betting platforms like Augur, and even virtual collectibles like NFTs (non-fungible tokens). Most interestingly, an ecosystem of on-chain decentralized finance (DeFi) emerged, with increasingly liquid capital markets accessible to anyone with a network connection. The security of all of these applications hinges on the endogenous trust Ethereum provides.

Unfortunately, Ethereum's endogenous trust suffers from similar problems compared to that of Bitcoin — trust-breaking governance and poor trust transference.

Complexity damages trust In its whitepaper, creator Vitalik Buterin articulated Ethereum's vision as a "simple" platform with "no features" based on a simple smart contract language (EVM) that "any average programmer can implement" [But+14a]. However, today Ethereum is an extremely complex system with a reference implementation of almost a million lines of code. [2] Such complexity demands constant maintenance and frequent changes. However, any consensus-breaking protocol updates (or "hard forks") in blockchains imply a substitution of endogenous trust for trust in the external protocol developer. As of June 2023, Table 2.2 lists every time the Ethereum protocol has "hard forked". [Eth]

As evidenced above, Ethereum's model of a complex dApp operating system necessitates frequent circumvention of endogenous trust.

[2] More precisely, around 1,000,000 lines of code, mostly in Go, are present in the Geth implementation. On the other hand, `btcd`, a Go implementation of Bitcoin, contains around 10 times fewer lines.

Table 2.2: History of Ethereum hard forks

Codename	Date	Reason
"Frontier"	Jul 30, 2015	*First release*
"Frontier Thawing"	Sep 7, 2015	*Introduction of "ice age"*
"Homestead"	Mar 14, 2016	*Protocol improvements*
"DAO Fork"	Jul 20, 2016	*Revert attack on DAO contract*
"Tangerine Whistle"	Oct 18, 2016	*Opcode reprice to prevent DoS*
"Spurious Dragon"	Nov 22, 2016	*Bugfixes and cost adjustments*
"Byzantium"	Oct 16, 2017	*Delay "ice age", new opcodes and features*
"St. Petersburg"	Feb 28, 2019	*Delay "ice age", new opcodes and features*
"Istanbul"	Dec 08, 2019	*New opcodes and features*
"Muir Glacier"	Jan 02, 2020	*Delay "ice age"*
"Berlin"	Apr 15, 2021	*Opcode reprice, more transaction types*
"London"	Aug 05, 2021	*Reform transaction fees, delay "ice age"*
"Altair"	Oct 27, 2021	*Add support for light clients to PoS "beacon chain"*
"Arrow Glacier"	Dec 9, 2021	*Delay "ice age"*
"Gray Glacier"	Jun 30, 2022	*Delay "ice age*
"Bellatrix"	Sep 6, 2022	*Prepare "beacon change" for Merge*
"Paris"	Sep 15, 2022	*"Merge"; switch to PoS*
"Shanghai"	Apr 12, 2023	*Allow PoS withdrawals*

Furthermore, Ethereum's developers and community perhaps do not all agree on the importance of endogenous trust. For instance, Vlad Zamfir, a core Ethereum developer, authored a well-known blog post arguing against "Szabo's law", which is the idea that blockchains should not be upgraded unless for critical, non-controversial reasons. In the same post, Zamfir also asserts that external intervention into a blockchain is a legitimate form of "crypto law".

Unsurprisingly, the combination of protocol complexity and social indifference towards endogenous trust has led to gross violations of endogenous trust that significantly disrupted the ecosystem. The most well-known example of this is the "DAO bailout" of 2016.

A case study: the DAO bailout On June 17, 2016, a vulnerability in the smart contract powering The DAO, one of the most popular Ethereum applications at that time, was maliciously exploited. The attacker stole an unprecedented amount of more than 3.6 million ETH — nearly 5% of all ETH in existence — from the contract.

The Ethereum community erupted into a heated debate about what to do about the attack. Eventually, an on-chain coin vote was held at short notice on July 15, 2016. 87% voted in favor of a hard fork to reverse the attack and reimburse investors in The DAO, though only 5.5% of all

outstanding ETH turned out at the vote. Four days later, a new version of Ethereum was released that executed the hard fork.

This highly controversial DAO fork completely subverted immutability, a crucial blockchain security property. A part of the Ethereum community continued to recognize Ethereum's original transaction history, forming a separate blockchain known as Ethereum Classic. As with Bitcoin's 2017 fork into Bitcoin and Bitcoin Cash, users ended up having to decide out-of-band which fork to trust, while application developers had to cope with the ground underneath their applications suddenly forking into two.

Ethereum's problems with trust transference By having more features that enable a richer dApp ecosystem, Ethereum sacrificed more *production* of endogenous trust than Bitcoin did. But does this buy better endogenous trust *transference* to application security?

In one sense, yes — unlike Bitcoin, Ethereum does support a wide variety of decentralized apps at a higher level of endogenous trust. For instance, trading on an exchange based on Ethereum smart contracts like Uniswap is categorically safer than doing so on something like Bisq.

Yet even this is insufficient, as nearly all dApp interaction uses intermediaries that are trusted third parties. DApps are typically accessed through a "frontend" webpage hosted on a traditional web host, and the JavaScript code in the frontend then uses trusted "RPC gateways" like Infura or Alchemy to interact with the blockchain.

These centralized providers do end up harming user security in similar ways to traditional service providers. For instance, in 2022 the famous Tornado Cash cryptocurrency tumbling smart contract was censored from mainstream RPC gateways due to an OFAC (Office of Foreign Asset Control) order [Coi22]. This prevented most users from accessing Tornado Cash — thus the on-chain security it provides cannot be transferred to users.

Other classical blockchains

Most other blockchains follow the same classical model of a global linear blockchain decided through Nakamoto consensus. Yet they nearly universally fall into the same taxonomy of Bitcoin-like application blockchains vs Ethereum-like platform blockchains, both suffering from serious application-blockchain friction that damages endogenous trust. Here we only discuss two of the most salient examples.

Namecoin Namecoin, launched in 2011, is arguably the earliest "altcoin", or non-Bitcoin blockchain. Like Bitcoin, Namecoin is an application blockchain, but it implements a secure and decentral-

ized DNS-like naming system rather than a mere cryptocurrency. Unfortunately, just like Bitcoin, it suffers from poor endogenous trust. As an application blockchain with a rather complex application, it hardcodes a great deal of application-level parameters like the starting price of domains into the protocol. These parameters turned out to be suboptimal — oversights in mechanism design caused problems like name squatting to be pervasive on Namecoin [Kal+15]. Tweaking these protocol rules has motivated several Namecoin "soft forks" that must be coordinated through exogenous trust. Furthermore, the combination of a small userbase and sharing the proof-of-work algorithm of Bitcoin seriously weakens Namecoin's Nakamoto consensus by making "51% attacks" against consensus exceptionally easy. In fact, successful attacks against Namecoin have compromised its security many times [Ali+16], and Namecoin relies even more on out-of-band coordination to prevent destruction of its security properties.

Ethereum Classic As the blockchain that forked from Ethereum "proper" by refusing to execute the DAO bailout, Ethereum Classic certainly aims to strictly enforce endogenous trust. Indeed, the Ethereum Classic community is well known to be strong proponents of "code is law". Yet this ideology has not prevented externally-coordinated protocol upgrades similar in number to those of Ethereum, often motivated likewise by application requirements. For example, the "Atlantis", "Agharta", and "Phoenix" protocol upgrades brought new features present in Ethereum to Ethereum Classic, to not fall behind in the set of features offered to applications. As of 2021, Ethereum Classic is somewhat ironically undergoing its own highly contentious ecosystem split between those in favor of fundamental protocol changes (like a new PoW algorithm, a protocol treasury, etc.) to improve functionality and proponents of strict immutability and endogenous trust — the latter publishing a document calling for "Ethereum Classic Classic". It is clear that despite the best of intentions, classical platform blockchains cannot avoid application-level entanglements that threaten endogenous trust.

2.3 Next-generation blockchains

Given classical blockchains' problems achieving endogenous trust, as well as their lack of many more surface-level goals like scalability and cost-efficiency, a consensus is developing that first-generation blockchains, both application and platform, are structurally problematic. Instead of producing yet another blockchain ruled by the "box" of application-blockchain friction, the blockchain trilemma, and similar difficulties, many newer systems completely rethink key design features of classical blockchains. Will this development hold the key to fulfilling the promise of reliable, transferable endogenous trust?

Projects that "miss the point"

Unfortunately, many next-generation blockchain projects do not address the problem of transferable endogenous trust. In fact, they often sacrifice it for other goals.

Quite a few projects, for instance, aim to massively parallelize transaction processing by completely abandoning the notion of collating transactions into a globally ordered and consistent chain of blocks. The most infamous example is probably IOTA, a cryptocurrency project that attempts to replace the blockchain with a "tangle" — an eventually consistent DAG of individual transactions, not subject to any kind of global consensus algorithm. This would supposedly allow greatly increased throughput for IoT (hence the name) and automotive applications. Unfortunately, the cryptoeconomics of IOTA's tangle are ill-defined and unexplored, and may not be sound [Ali+19]. In practice, the actual deployed IOTA network has relied for many years on a centralized "coordinator" to ensure consensus, rather than the IOTA tangle [Fou22]. In abandoning blockchains in pursuit of a maximally scalable transaction broadcast network, IOTA unfortunately also abandons what makes blockchains interesting in the first place: endogenous trust.

Another class of newer blockchains takes aim at Nakamoto consensus, which is certainly a source of problems ranging from very poor energy efficiency (due to the proof-of-work mechanism) to the lack of any finality (there's always the possibility that the canonical chain suddenly becomes non-canonical). Unfortunately, many of the proposed solutions also ruin the (relatively) sound cryptoeconomics of Nakamoto consensus. For example, on "delegated proof-of-stake" (dPoS) blockchains like Eos, DASH, and Cardano, cryptocurrency holders elect a small committee of consensus nodes. The blockchain history is then decided entirely through a Byzantine fault-tolerant consensus algorithm, like PBFT or Tendermint, run among these nodes. DPoS does achieve drastically higher transaction throughput than Nakamoto consensus, but does so by drastically weakening the cryptoeconomic incentives that underpin endogenous trust — in dPoS, there is no endogenous mechanism to incentivize voting in "good" nodes. Unsurprisingly, we see an array of pathological behavior in practice, such as vote buying and even nationalistic appeals emerging during Eos consensus node elections. Once elected, Eos consensus nodes often engaged in security-violating actions such as freezing on-chain accounts.

On-chain governance

In contrast to projects like IOTA and Eos, a small but particularly interesting category of newer-generation blockchains does attempt to fix the dubious security of ad-hoc, externally-coordinated upgrades through *on-chain governance*. In these blockchains, periodic changes to consensus rules are accepted as a fact of life, but protocol updates are developed in a decentralized and open manner and confirmed by an unforgeable, on-chain vote.

As of 2021, Tezos is probably the most mature blockchain in existence that implements on-chain governance. Tezos experiences an "election cycle" every 6 months, during which proposals to upgrade the protocol are discussed and refined. At the end of each cycle, the blockchain protocol *self-amends* according to the proposals submitted and voted on-chain in the cycle. This allows Tezos, a blockchain comparable in complexity to Ethereum, to gracefully coordinate protocol upgrades while avoiding contentious community splits or suddenly breaking blockchain applications that assume an immutable blockchain.

Although on-chain governance is a fascinating innovation that may very well be the best way of guiding the development of a complex and ever-evolving blockchain like Tezos, it nevertheless fails to improve the state of endogenous trust in blockchains. Even though protocol changes in a blockchain with on-chain governance are coordinated internally, the process of developing and approving the changes involves humans and thus is necessarily off-chain. Unless a blockchain can autonomously derive how to improve itself and incentivize its participants to apply those improvements, any significant protocol upgrade will involve compromising endogenous trust.

Minimal, "data availability" blockchains

A final category of next-generation blockchain that merits discussion is minimal "data availability" blockchains, like Celestia (formerly known as LazyLedger). These blockchains generally do not have higher-level semantic features like smart contracts, coin scripts, etc., and instead, only collate transactions that may contain arbitrary data into blocks. The interpretation of these transactions is left entirely to the application, with the blockchain serving as a mere broadcast bus. Due to their extremely simple functionality, data availability blockchains have the potential to achieve very strong endogenous trust. Lack of complex features makes cryptoeconomic mechanism design easier, while also greatly reducing the prospect of interventionist governance and disruptive upgrades: there's not much in the protocol to change!

Unfortunately, blockchains without semantics cripple the *transferability* of endogenous trust. Building something like a full-featured cryptocurrency transfer system on a data-availability blockchain will involve almost as much complexity as a Bitcoin-like blockchain, as well as the rigid application-protocol coupling of Bitcoin. The cryptocurrency application as a whole will not really inherit the strong endogenous trust of the underlying blockchain, since it does not have enough functionality to transfer its security to upper-level protocols. This is a point also alluded to by Vitalik Buterin in a blog post [But19a], where he argues that layer-1 blockchains at least need some features for applications to be able to root their security entirely in the blockchain's endogenous trust and achieve "functionality escape velocity".

2.4 What should we do about it?

So, existing blockchains aim and fail at transferable endogenous trust. The natural question, then, is how can one build a blockchain that attains this holy grail? To do so, one must avoid the problems crippling transferable endogenous trust in existing blockchains. These problems have two main sources.

First, *weak cryptoeconomics* often undermine incentive structures intended to secure the blockchain endogenously and force the community to resort to out-of-band coordination to keep the ledger secure. Unpredictable social processes such as Bitcoin's coordination surrounding SPV mining and EOS's node elections all result from a lack of built-in incentives nudging participants towards secure behavior.

The second, and perhaps more important, cause of poor endogenous trust is *application-blockchain friction*. Application-blockchain friction occurs when a blockchain protocol becomes increasingly unsuitable for its main application. When this happens, the blockchain loses its users' confidence. This forces a contentious out-of-band protocol upgrade to prevent the blockchain from passing into the dustbin of history. The Bitcoin block-size controversy is the most well-known case of loss of trust from application-blockchain friction — unsurprising given Bitcoin's rigid coupling of its core payment application to its blockchain.

Unfortunately, general-purpose blockchains like Ethereum experience even more challenges to their endogenous trust due to application-blockchain friction. In fact, as we have seen in the previous section, most of the protocol upgrades to Ethereum so far involved fairly minor tweaks to functionality in order to support newly emerging, unanticipated applications.

The prevalence of application-blockchain friction is because both application blockchains (like Bitcoin) and platform blockchains (such as Ethereum) *are on the wrong protocol layer*. Both are *too close to applications*.

Platform blockchains like Ethereum sit directly underneath applications to allow apps' easy deployment — a new cryptocurrency can be implemented on Ethereum in a few dozen lines of code. Such a direct interface between application and blockchain, however, inevitably results in contention between ever-changing application requirements and ideally immutable blockchain protocols. The situation is analogous to telecommunication networks before the Internet, where a vertically integrated system directly offered relatively high-level functions like voice calling and teletype. Like Ethereum, these complex platforms were extremely costly to upgrade when they were forced to change by the rise of new applications and technological advances.

A final pitfall is *poor trust transference*, where the blockchain itself has endogenous trust but applications that use the blockchain find it hard to gain the same trust. This problem prevents us from achieving useful endogenous trust simply through having the blockchain do as little as possible.

Learning from previous mistakes, it is clear that a blockchain with transferable endogenous trust must first be built upon a solid cryptoeconomic foundation. This ensures that its core security properties will require no out-of-band social coordination to uphold. More importantly, it must also minimize application-blockchain friction and trust transference by using a design philosophy analogous to that of the Internet Protocol — a small, low-level protocol with straightforward semantics, yet enough semantics to generalize to a vast field of applications, combined with powerful light clients that allow non-blockchain code to easily access these semantics without losing security. In this architecture, flexible "middleware" protocols to separate the blockchain from the ever-changing needs of user-facing functionality, and pervasive use of blockchain light clients cement the security of these protocols on that of the blockchain. Thus, the blockchain can remain an embedded "endogenous trust engine" for decades without changing.

Themelio is the new blockchain designed to embody that vision.

Chapter 3

Themelio: a neutrality-maximizing blockchain

Earlier, we outlined transferable endogenous trust as the core goal of public blockchains that sets them apart from other computer networks. In this part of the thesis, we describe Themelio, a blockchain designed to maximize transferable endogenous trust.

We begin by introducing the concept of *neutrality-maximizing abstraction* — a "Goldilocks zone" of abstraction that is low-level enough to decouple from user-facing applications but high-level enough to decouple from concrete implementation strategies. We use the Internet Protocol (IP) as a case study.

3.1 Neutrality-maximizing abstraction

Case study: the Internet Protocol

It is undeniable that the Internet Protocol (IP), which underlies all Internet transmission, is the most successful telecommunication protocol of all time. But it is somewhat less appreciated that it is unlike most modern telecommunication systems in design: IP specifies neither how it is to be implemented nor how user-facing applications are supposed to use it. Instead, it focuses on doing one highly general but simple task — best-effort routing of datagrams. This stands in contrast to pre-Internet telecom protocols, which were tightly coupled to and carefully engineered for applications like telephone and television.

Yet IP's unreliable datagrams turned out to be a neutrality-maximizing abstraction. On one hand, any physical medium that digital signals can be modulated over can carry IP packets. On the other hand, with the help of intervening protocols like TCP and UDP, nearly any telecommunication application can be efficiently expressed in terms of point-to-point routing of unreliable datagrams.[1] This means that both physical media and applications can freely evolve without any changes in the IP protocol itself.

In practice, IP turned out to be extremely stable — IPv4, the most common version of IP deployed today, was invented in the 1980s. This contrasts with telecom protocols that lack neutrality-maximizing abstraction, which tend to stifle innovation. In those protocols, even small application-level innovations, like pagers, involved governance-intensive upgrade efforts reminiscent of blockchain protocol upgrades.

An important point to note is that IP's neutrality-maximizing abstraction not only eliminates much of the technical need for application-driven protocol upgrades, it encourages *social* resistance to protocol change. Due to IP's unique abstraction level, IP implementations end up becoming deeply embedded in extremely diverse protocol stacks, all of which assume that IP behaves a certain way and cannot tolerate changes. Nowhere is this phenomenon more clear than in the difficulty that networks face switching from IPv4 to IPv6. Ever since IPv6 was created in 1998, many people have spent great efforts trying to replace the older IPv4 with its technologically superior successor. Impressive-sounding events, like the Internet Society's "World IPv6 Day" in 2011 and "World IPv6 Launch Day" in 2012 were supposed to jump-start IPv6 adoption. But IPv4, despite severe technical shortcomings like a drastic shortage of IP addresses, is still around. By 2022, only a little more than 30% of networks support IPv6, and the vast majority of Internet traffic continues to be IPv4-based.

Neutrality-maximizing abstraction in blockchains

So we see that given an entrenched neutrality-maximizing protocol, protocol changes cannot get executed, even with widespread support and essentially zero opposition. Though in IP's case it hampers important technical improvements, this sort of protocol stubbornness turns out to be crucial to endogenous trust in blockchains. Protocol upgrades or "governance" in general require external, preexisting trust in whatever social mechanism decides the upgrades. This is because the entire endogenous trust of blockchains is founded upon cryptoeconomic mechanisms inherent in the protocol. Thus, endogenous trust crucially depends on actually being able to pin down what cryptoeconomic mechanisms will be in force! Alice and Bob trust the smart contract that intermediarizes their transaction only because they know that the contract code is immutable.

[1] Two exceptions may be latency-critical and inherently broadcast applications, where circuit-routed telephone and broadcast radio still retain niches

Merchants accept bitcoins for the goods they sell only when they are confident that tomorrow some protocol change wouldn't freeze all their bitcoins.

We also see that all the existing blockchains that lack strong, transferable endogenous trust lack neutrality-maximizing abstraction. Application and platform blockchains alike have abstractions close to the application, analogous to the Google Searches and GraphQLs of the Internet, while data-availability blockchains have excessively low-level abstractions analogous to physical transmission media.

3.2 Goals and principles

The overarching goal of Themelio is, therefore, to construct a blockchain with *neutrality-maximizing abstraction*. Concretely, this manifests in three principal design principles: *protocol simplicity*, *cryptoeconomic robustness*, and *layered scalability*.

3.2.1 Protocol simplicity

Protocol simplicity is arguably the most important design principle of Themelio, and it includes two facets — implementation and interface. On one hand, we want Themelio to have a simple specification that is easily implementable in most programming languages, with elegant mechanisms with clear rationales. On the other hand, we want to maximize the simplicity of the interface between Themelio and upper-layer protocols or applications, avoiding leaky abstractions

These two goals are occasionally in tension, but they are for the same purpose: reducing or eliminating the need for protocol revision by zooming into the neutrality-maximizing level of abstraction. More "moving parts" in a protocol increase the chances of something not being right in the first revision — for example, Ethereum has tweaked the gas costs of EVM opcodes through consensus-breaking updates multiple times in its history. It also makes it harder to ensure bug-free implementations, and fixing bugs is another reason for constant maintenance.

Leaky abstractions can have "contagious" effects that force technical quirks of the blockchain to entire ecosystems — an example is how the lack of finality in Nakamoto consensus robs finality from all blockchain applications, forcing them to consider "time travel" as a possibility.[2] Such quirks often require applications to give up entirely relying on the blockchain's endogenous trust in exchange for more reasonable behavior. Furthermore, contagiously leaky abstractions' coupling of applications to

[2] For example, this makes it unsound to couple an off-chain action, like allowing someone access to a server, with on-chain transactions, unless the off-chain action is reversible

the blockchain introduces the specter of endogenous-trust-harming application-blockchain friction. This can lead to a great deal of competing pressure to change the protocol by users harmed by buggy applications: see, for example, the controversies surrounding the "DAO hack" hard fork of Ethereum or the block-size hard fork of Bitcoin.

3.2.2 Cryptoeconomic robustness

As the ultimate backstop of endogenous trust, robust and lasting cryptoeconomic mechanisms must be used to incentivize "correct" behavior from untrusted participants, while external, community-based coordination cannot be relied upon. Thus, we avoid "ticking time bomb" mechanisms such as Bitcoin's continually decaying block reward or Ethereum's ice age, as they essentially leave the long-term security of the system up to future governance decisions. We also avoid honest majority and similar assumptions that model attacks by malicious participants as a fault-tolerance problem, as we believe that absent a strong cryptoeconomic mechanism to incentivize honest behavior, making judgments about the proportion of "bad guys" is exogenous rather than endogenous trust. We take a generally conservative approach to threat models — for example, we usually assume that collusion to achieve malicious goals is possible, but not altruistic collusion to defend the system.

Finally, cryptocurrency price volatility is an often neglected cause of instability in cryptoeconomic mechanisms. Thus, Themelio uses a novel trustless value-stable cryptocurrency as its built-in unit of account. A stable unit of account strengthens mechanism design in two important ways. First, it allows for the extensive use of fixed and slowly-changing protocol fees. Second, it makes many simplifying assumptions like "rational actors want to maximize the number of cryptocurrency units they own" much closer to reality.

3.2.3 Layered scalability

The final pillar of Themelio's design is layered scalability. This means that we avoid trying to scale applications that inherit Themelio's security squeezing as much functionality and performance into Themelio itself as possible, as that leads to application-blockchain coupling and the exceptional engineering challenges of designing a scalable, strongly consistent, global broadcast network. Instead, we design Themelio with an interface that maximizes trust transference to separate, horizontally scalable "layer 2" protocols that do not require global broadcast.[3] This implies that though we do want to prevent the blockchain from being such a performance bottleneck that it hinders applications that truly require global broadcast, we refrain from techniques such as sharding that greatly increase

[3]For example, an on-chain naming system may serve as the PKI for a fully off-chain secure communication standard

complexity, or techniques such as massively increasing block size that inherently weaken endogenous trust.

Thus, we design Themelio to have the highest attainable throughput given a globally consistent, non-sharded data model, while supporting features such as powerful trustless light clients and state-channel-friendly scripting that allow non-blockchain logic to fully inherit blockchain-based trust. We expect most Themelio applications to consist of a small amount of rarely changing on-chain state (such as the channel graph in a payment channel network), combined with horizontally scaling, logically decentralized off-chain state.

In fact, this sort of layered scalability is not just crucial for performance, but also trust transference. Since humans do not live on-chain, no application can be fully on-chain, and every use of blockchains must deal with ensuring that some off-chain program (such as a DeFi frontend) inherits the security of the blockchain. Designing for the off-chain consumption of Themelio's security by non-blockchain protocols would solve this problem as well.

3.3 Blocks, transactions, and state

In this section, we discuss Themelio's abstract model of the state of the blockchain. Themelio uses a richly-scripted coin-based model that is different from both the simple coin-based model of Bitcoin and the accounts-and-contracts model of Ethereum.

3.3.1 Background and conventions

Blockchains as state machines

Throughout this section, we will be discussing Themelio as a *transaction-based state machine*, a conceptual framework introduced by the creators of Ethereum. What this means is that a blockchain starts with a genesis state, which is mutated by transactions over time, finally ending at the blockchain at its current state. *Transactions*, in this model, are arcs between valid states, and only states that can be reached by repeatedly applying valid transactions to the genesis state are valid states in the blockchain. Formally,

$$\sigma_{i+1} \equiv \Upsilon(\sigma_i, T) \tag{3.1}$$

where Υ is the state transition function and T is a transaction.

A blockchain, however, does not exactly consist of a linear history of transactions — transactions are collated into *blocks*, each of which can contain a large number of transactions, and possibly

auxiliary data. This collation process then forms a coarse-grained journal of transactions. A more accurate formalization, then, uses a *block-level* state transition function Π:

$$\sigma_{i+1} \equiv \Pi(\sigma_i, B) \tag{3.2}$$

$$B \equiv (\dots, \{T_0, T_1, \dots\}, \dots) \tag{3.3}$$

$$\Pi(\sigma, B) \equiv \Omega(B, \Upsilon(\Upsilon(\sigma, T_0), T_1) \dots) \tag{3.4}$$

where Ω is the *block finalization function* that assembles a final state out of the result of applying all the transactions T_i within block B. Note that the block is an *unordered* set of transactions, and Υ can be applied to the transactions within a block in an arbitrary order.

We believe that treating a blockchain as fundamentally a state machine is a much more helpful model than simply a series of transactions. In fact, Themelio extensively uses explicit on-chain representations of the current blockchain state, rather than making most of the state implicit as in Bitcoin. This actually follows more closely the practice of Ethereum-type state-based blockchains.

Cryptographic primitives

Themelio uses a number of cryptographic primitives throughout the system:

- $H(x)$ denotes BLAKE3 [OCo+20], a 256-bit secure hash function.
- $\mathsf{Sig}(m, \mathrm{sk})$ and $\mathsf{EdVer}(m, \mathrm{sig}, \mathrm{pk})$ respectively denote signing and verifying Ed25519 [MJ15] signatures.

3.4 Coins, not accounts

Before we get into the technical details of Themelio's data structures, we explore a high-level principle of Themelio's state — its choice of a Bitcoin-like "coin-based" state rather than an Ethereum-like "account-based" state. The only key differences from Bitcoin's are two: the entire coin and transaction graph is indexed into sparse Merkle trees, and coins can be locked by arbitrary scripts.

Such a model is unusual for a general-purpose blockchain, but we believe it has significant advantages over account-based models for the minimal root of trust we are building.

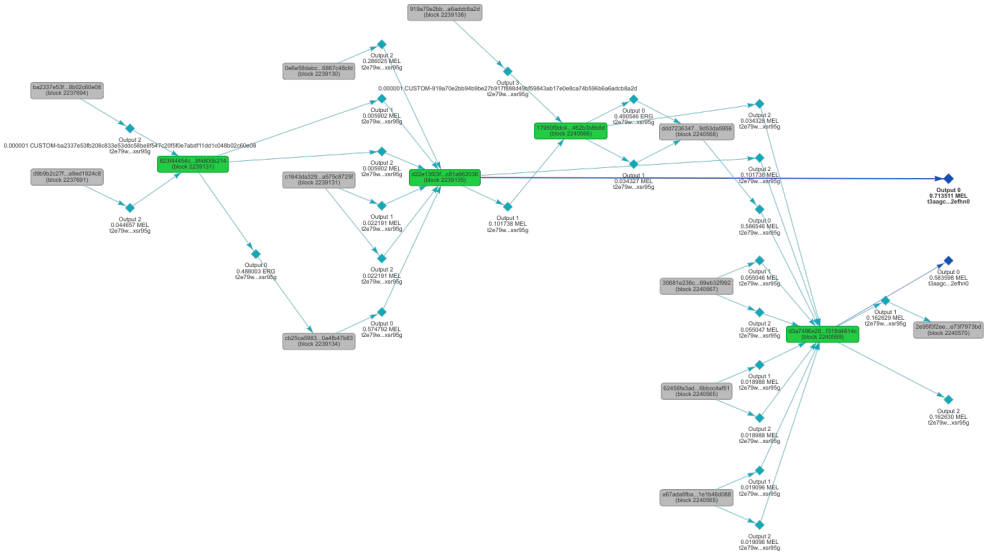


Figure 3.1: A fragment of the Themelio transaction graph

3.4.1 Coin-based transactions

Themelio's basic transaction model belongs to a family usually known as "UTXO-based" or "coin-based" models. This is the oldest family of blockchain models, including first-generation blockchains like Bitcoin and Litecoin. In a coin-based model, the blockchain can be understood as a grow-only directed acyclic graph (DAG) of *transactions*. Every transaction on the blockchain takes as input and spends one or more *unspent transaction outputs* (*UTXOs*) of previous transactions, which are informally known as *coins*. It then produces as output one or more coins that can be spent as input by subsequent transactions.

Every coin represents a given amount of cryptocurrency, known as its *value*, and it includes an *unlock constraint* that specifies what sort of transaction can spend the coin. Each coin can only be spent once. Excepting transactions that "mine" more currency, the sum of the values of all the coins spent by a transaction must equal the sum of the values of all the coins created by it.

Translating this into the formal state-transition model, this means that the current state σ consists of a set of coins with corresponding constraints and values, while the per-transaction state transition function Υ atomically removes and adds new coins, without changing the state in any other way.

Let us illustrate how coin-based transactions work with a simple example. Assume there are 5 coins identified as $B_1, \ldots, B_5$, each worth \$1, and each having an unlock constraint specifying "any transaction that spends me must have Bob's signature". Informally, we say that Bob owns 5 coins,

each one worth $1. Bob "owning" a coin simply means Bob knows how to satisfy the coin's unlock constraint.

Now, assume that Bob wants to send his friend Alice $2.5. He creates a new transaction spending B_1, B_2, B_3 as input, with two outputs:

- A_1 with value $2.5 and a constraint requiring Alice's signature.
- B_6 with value $0.5 and a constraint requiring Bob's signature.

and informs Alice about A_1. Bob now "owns" B_4, B_5, B_6 with a total value of $2.5, and Alice owns A_1 with a total value of $2.5, just as we wanted. Note that Bob had to give himself a new coin for the transaction to balance; this new coin is known as a *change output*. Figure 3.1, generated by the Themelio block explorer Melscan [Mel23], shows a complex series of interdependent coin-based transactions.

3.4.2 Why coins?

In Themelio, we use coin-based transactions with a cryptocurrency that we call the *mel*. We believe that a model of interdependent transactions spending and producing coins, though originally invented only for modeling money transfers, is a very good abstraction on which endogenous-trust applications can be built.

But coin-based models are not popular at all among general-purpose blockchains. Most blockchains attempting to support general decentralized apps use *account and smart-contract* based models, like that of Ethereum [Woo14]. In these models, accounts directly map to sums of money that can be transferred and accounts can have automatically executing code attached. In fact, even Qtum [DMN18], one of the only coin-based general-purpose blockchains, uses an "Account Abstraction Layer" to simulate Ethereum-like accounts. Why do we believe coins are the way to go?

Firstly, coins allow Themelio to *process transactions quickly*. In an account-based model, like in Ethereum or traditional banking, strict global transaction ordering is necessary. Yet coin-based transactions can be processed in any topological order — we simply need to process the transaction that produces a coin before the transaction that spends it. Transactions within a block can be validated mostly in parallel. This greatly increases performance.

Secondly, a coin-based architecture *simplifies state transitions*. Blockchain protocols can be thought of as state-transition functions, where each transaction takes in the "world" in a certain state (say, Bob having $5 and Alice $0) and outputs a different state (Alice and Bob both having $2.5). To support functionality beyond basic payments, account-based blockchains like Ethereum

need arbitrarily mutable global state, accessed by user-programmable "smart contracts". However, programming decentralized apps with mutable state is notoriously prone to error. Complex state transitions are associated with difficult-to-find bugs and blockchain-level performance problems. In a coin-based blockchain, state is extremely simple: the set of all unspent coins. All transitions simply correspond to individual transactions deleting and adding coins atomically. This leads to clearer logic in decentralized apps and faster performance.

Finally, coin-based transactions are *highly expressive and light-client legible*. A very large class of security-critical problems boils down to establishing a consistent, valid graph of interdependent events. For example, in a naming system, a successful name transfer depends on previous events like the previous owner relinquishing control, that owner first registering the name, and so forth. Centralized roots of trust, like notaries, certificate authorities, and banks, almost always serve the role of ensuring consistency of an event graph. In a coin-based blockchain model, the transaction DAG maps extremely well to these event graphs. When combined with the trustless light clients enabled through extensive Merkle indexing, Themelio's coin-based model enables easy trust transference to decentralized protocols and apps.

3.4.3 Rich covenant scripting

In order to support "coin-oriented" on-chain constructions, Themelio unlock constraints, called *covenants*, use a simple yet powerful scripting language: Melodeon. Unlike Bitcoin unlock scripts, Melodeon can place conditions on any part of the transaction attempting to spend a coin and enables easy development of a wide variety of decentralized apps. Yet unlike Ethereum's EVM, Melodeon has no access to persistent state, eliminating a large class of "smart contract" bugs. Melodeon's programming model is vaguely similar to what one will get if Bitcoin accepted the "Bitcoin covenants" [OP17] proposal, as well as giving scripts general computation ability and an Ethereum-like "gas" model. Melodeon is a *bounded loop* language, stopping short of Turing completeness but able to compute any primitive recursive function. [4]

Melodeon is written in a vaguely ML-like syntax and compiled to a stack-based bytecode to be embedded in transaction outputs. Simple, Bitcoin-like constraints are straightforward. For example, the following is Melodeon for a "multisignature" constraint, for coins requiring signatures from both `ALICE-KEY` and `BOB-KEY` to be spent:

```
// Alice's signature in slot 10, Bob's in slot 11
ed25519_signed_by(alice_public_key, 10) &&
```

[4] Interestingly, the Turing-machine state transition function itself can be computed by a bounded-loop language, so Melodeon covenants can in fact calculate anything over multiple invocations, as long as it is suitably broken down into bounded chunks. So in practice, Melodeon is used as a general programming language.

Table 3.1: Comparison of EVM, Bitcoin Scripts, and Melodeon features

Feature	EVM	Bitcoin Scripts	Melodeon
Turing-complete	Yes	No	Almost (BLOOP)
Data model	Stateful accounts	Stateless coin unlock	Stateless coin unlock
Gas model	Yes	No	Yes

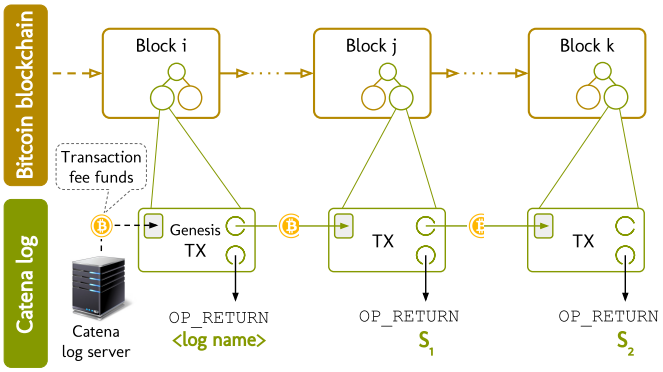


Figure 3.2: Example of a Catena log

```
ed25519_signed_by(bob_public_key, 11)
```

The most useful covenants in Melodeon are not simple filters like this, but covenants that constrain covenants. This allows us to embed a wide variety of decentralized, permissionless secure data structures within the transaction graph, which we might call *coin structures*.

This is confusing, so let us illustrate the concept with an example. Catena [TD17] is an append-only log originally implemented in Bitcoin. The basic idea is simple: a central authority can transparently publish a log of messages by building a transaction chain, each spending the first output coin of the previous transaction. Since coins cannot be spent twice, the authority cannot rewrite, reorder, or delete any log entries after they are published. Figure 3.2 is an example of a Catena log.

In Bitcoin and other existing coin-based blockchains, Catena logs must be maintained by central authorities. Coins forming the chain must be "owned" by the log publisher, lest someone spend them for other purposes and ruin the log. This prevents the use of Catena in applications without a central publisher. In Themelio, however, we can easily write a Melodeon covenant that only allows transactions that grow the Catena chain to spend the coin. Any coin with the following covenant is forced to be the start of a permissionless Catena chain that can never be broken:

```
// there must be a first output
```

```
// first output must be constrained by a covenant
// with the same hash as this program
let first_output = vget(env_spender_tx().outputs, 0) in
first_output && first_output.covhash == env_self_hash()
```

Coin structures, of course, are not limited to simple logs. Bitforest [DKB18] builds an entire naming system out of a coin structure that implements an equivocation-proof binary search tree, yet like Catena it must rely on a centralized coin owner when deployed on existing blockchains. Analogous Melodeon constraints can be used to implement Bitforest on Themelio as an entirely decentralized and permissionless naming system, with features comparable to naming systems on "smart contract" blockchains, like the Ethereum Naming System (ENS).

3.5 Block and transaction structure

We now discuss the structure of blocks and transactions in Themelio.

3.5.1 A note on notation

We introduced the notion of seeing the blockchain in terms of state transitions using mathematical notation akin to that used in existing work such as the Ethereum yellow paper. However, standard mathematical notation poses an unnecessary obstacle to clarity in two common situations:

- There is no obvious way of representing data with many named fields ("structs"). Sets of key-value tuples do not capture the fact that a canonical field ordering exists, while straight tuples of values force the keys of a struct s to be the extremely inconvenient s_1,s_2, etc.
- When we need to introduce many variables, the mathematical convention of using single-letter variables hinders clarity just like code that uses single-letter variables. It is often difficult to keep track of a bunch of uppercase, lowercase, and Greek letters that are not shorthand for obvious words.

Thus, for the remainder of the paper, we will use a "hybrid" approach:

- Actions will be written in terms of both mathematical functions and pseudocode
- Datatypes will be defined using the Rust programming language

- Names are often "codelike", i.e. we might name a transaction foobar rather than T_1.

We hope this combines the clarity of "pseudocode notation" and the succinctness of mathematical notation.

3.5.2 State transitions at a glance

Let us first take a look at the basic flow of Themelio's state transitions. As the following picture illustrates, this is based on two basic datatypes `State` and `SealedState`. Both of these datatypes will be described in further detail later.

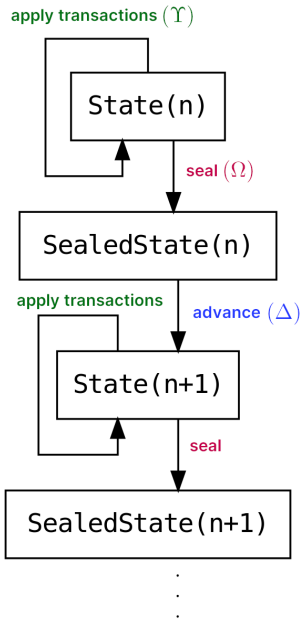


Figure 3.3: State transition flowchart

`State`: transaction-level state

State: transaction-level state

`State` is the datatype representing the transaction-level state in Themelio. Transactions can be **applied** to `State`s using the state-transition function Υ.

However, transaction-level state transitions cannot create new blocks. With only Υ, each `State` is "stuck" at a particular height; there is no transaction that can force the blockchain to grow to a new height.

Thus, Υ can also be considered the *intraheight* state transition function, and transactions intra-height state transitions. `State` can also be thought of as representing the incomplete *next* block that is being built.

When a block is ready, the `State` is **sealed** with the block sealing function Ω to produce a `SealedState`.

`SealedState`: block-level state

`SealedState` represents the block-level state in Themelio, and corresponds to a certain block height. It is "sealed" in the sense that no more transactions at that block height can be accepted. `SealedStates` are the subject of blockchain consensus, and the canonical blockchain state is defined only through a series of `SealedStates`.

The only operation possible on a `SealedState` is to **advance** it, converting it to a `State` that accepts transactions for the next block.

3.5.3 Common functions & datatypes

We now discuss some supporting functions and datatypes that will be recurring in our description of Themelio's logic:

Serialization

Themelio uses `bincode`[5] serialization. Bincode is very fast and well-integrated with Rust's `serde` serialization ecosystem, which Themelio's reference implementation extensively uses. More importantly, it is a *canonical* serialization: each serializable object has only one canonical serialization, in contrast with formats like JSON. This makes concepts like "the hash of a transaction" trivially well-defined.

Bincode is a family of encoding formats; we use the variant with little-endian, variable-length encoding of integers and disallow trailing bytes.

We omit serialization and deserialization from our algorithm descriptions. For example, the hash of the bincode serialization of v is simply denoted $H(v)$.

[5] `https://crates.io/crates/bincode`

Cryptography

- `HashVal` represents a 256-bit hash, generally an output of the BLAKE3-256 hash function. The BLAKE3-256 hash function of a value v is denoted $H(v)$, while the BLAKE3-256 keyed hash with key k is denoted $H_k(v)$.
- `Ed25519PK` represents an ed25519 public key.
- `Ed25519SK` represents an ed25519 secret key.

Sparse Merkle trees

Key-value mappings in the Themelio state are generally represented as *sparse Merkle trees* [DPP16]. Sparse Merkle trees use some neat tricks to efficiently encode a Merkle tree with 2^{256} elements, which can be used as a mapping from 256-bit keys to values. An important property is that given any SMT of N elements, there is a 256-bit **root hash** that uniquely identifies the dictionary, and **proofs of inclusion and exclusion** of size $\Theta(\log N)$ can be produced. Anybody with the root hash can use these proofs to verify that a certain key-value binding either exists or does not exist in the SMT.

In our discussion, a *typed* SMT between datatypes `K` and `V`, `SmtMapping<K, V>`, is often used. This denotes an SMT mapping between $H(k), k \in K$ and values of type V. Given a value M of type `SmtMapping<K,V>`, we also write

- $v = M[k]$ to denote the value mapped to by k
- $\pi = \mathsf{Pie}(M, k)$ to denote a **p**roof of **i**n/**e**xclusion that the key k either maps to some $v = M[k]$, or nothing at all

SMTs allow us to commit to large key-value datasets succinctly, and are key to thin-client scalability in Themelio.

3.5.4 World state

Overview of elements

The *world state*, `State`, is the basic structure that encapsulates all the information needed to validate a single transaction. There is a one-to-one mapping between world states and entire blockchain histories, and other concepts such as the block-level `SealedState` are derived from `State`.

`State` is defined as follows:

```
pub enum NetID {
    Testnet = 0x01,
    Mainnet = 0xff,
}

pub struct State {
    // Identifies the network.
    pub network: NetID

    // Core state
    pub height: u64,
    pub history: SmtMapping<u64, Header>,
    pub coins: SmtMapping<txn::CoinID, txn::CoinDataHeight>,
    pub transactions: SmtMapping<HashVal, txn::Transaction>,

    // Fee economy state
    pub fee_pool: u128,
    pub fee_multiplier: u128,
    pub tips: u128,

    // Melmint/Melswap state
    pub dosc_speed: u128,
    pub pools: PoolMapping,

    // Consensus state
    pub stakes: SmtMapping<HashVal, StakeDoc>,
}
```

We now take a look at its individual elements.

Core state

Network ID The state always carries a *network ID*, which identifies whether the state belongs to the canonical "mainnet" or a temporary "testnet". The genesis state has a hardcoded network ID, and this ID can never be changed by any state transition. This makes different networks separate at the level of the state-transition function.

Height and history The first two elements position the state within the series of blocks that form the blockchain:

- `State::height` is the number of blocks since the beginning of the blockchain. Thus, we talk about the block with height 0, 1, 2, …
- `State::history` is an SMT that maps each *previous* height with a `Header`. `Header` is a fixed-size type that summarizes and commits to `State`:

```
pub struct Header {
    pub network: NetID,
    pub previous: HashVal,
    pub height: u64,
    pub history_hash: HashVal,
    pub coins_hash: HashVal,
    pub transactions_hash: HashVal,
    pub fee_pool: u128,
    pub fee_multiplier: u128,
    pub dosc_speed: u128,
    pub pools_hash: HashVal,
    pub stake_doc_hash: HashVal,
}
```

Coin mapping The most important component of Themelio's world state is the coin mapping, `State::coins`. Each key in this SMT is a `CoinID`, a structure that uniquely identifies a coin by the hash of the transaction that produced it, as well as an index into its outputs:

```
pub struct CoinID {
    pub txhash: HashVal,
    pub index: u8,
}
```

For example, `CoinID{txhash: foobar, index: 1}` identifies the second output of the past transaction with hash `foobar`. Each `CoinID` maps to a `CoinDataHeight`:

```
pub struct CoinData {
    pub covhash: HashVal,
```

```
    pub value: u64,
    pub denom: Vec<u8>,
}

pub struct CoinDataHeight {
    pub coin_data: CoinData,
    pub height: u64,
}
```

This essentially encapsulates the transaction's "associated data". More specifically:

- `covhash` specifies the hash of the Melodeon covenant that constrains the transactions allowed to unlock this coin
- `value` specifies the value of the transaction
- `denom` identifies the denomination of the coin. Generally, this is the hash of the transaction that first created the new denomination. There are three special cases for builtin assets:
  - `m` identifies micromels
  - `s` identifies microsyms
  - `d` identifies microdosc

The basic action of a transaction is to remove coins from `State::coins` and put newly created coins in.

Transaction mapping The transaction mapping contains all the transactions within the last block, mapping the transaction hash $H(T)$ to the transaction T. Transactions themselves are structures which we will describe in a later section.

Note that the world state does not anywhere contain the ordering of the transactions. This is because **transactions within a block are unordered**: unlike almost all existing blockchains, there is no defined order of transactions beyond which block they belong to. This property allows for easy parallelization of transaction processing — a property also found in blockchains like Solana [Sol23] and first formulated for UTXO blockchains by a Bitcoin Cash proposal [Ver+18], but not Bitcoin or Ethereum.

Fee economy state

The fee economy state consists of:

- `State::fee_pool`, the **fee pool** of accumulated base fees that fund staker rewards. This can be thought of as belonging to all stakers.
- `State::fee_multiplier`, the **fee multiplier** that scales the amount of fees transactions are required to have.
- `State::tips`, the **tips** that are fees local to this block, paid to the block proposer when this `State` is sealed.

These variables interact with Themelio's fee system, which we will describe in the transaction-level and block-level state transition functions.

Melmint/Melswap state

The Melmint/Melswap state is used to control the Melmint mechanism that stabilizes the value of each mel. This consists of:

- `State::dosc_speed`, the **DOSC speed** that measures how much work the fastest processor can do in 24 hours.
- `State::pools`, a mapping from token denominations to values of type `PoolState`

The precise ways these variables are used are discussed in the Melmint/Melswap specification.

3.5.5 Consensus state

The consensus state is used to keep track of the stakers that participate in Themelio's consensus. For every transaction T that stakes a certain number of syms, it maps $H(T)$ to a value of type `StakeDoc`:

```
pub struct StakeDoc {
    pub pubkey: Ed25519PK,
    pub e_start: u64,
    pub e_post_end: u64,
    pub syms_staked: u64,
}
```

The fields of `StakeDoc` have the following significance:

- `pubkey` denotes the ed25519 public key of the staker, used in consensus
- `e_start` denotes the first **epoch** (period of 100,000 blocks) that the staker has voting power. This must be greater than the epoch in which T was confirmed.
- `e_post_end` denotes the first epoch *after* the last epoch in which the staker has voting power. For example, if a staker has voting power for epochs 1, 2, and 3, then `e_post_end` is 4.
- `syms_staked` denotes the number of syms staked by this staker.

The details of how these values are used will be discussed in the state transition functions.

3.5.6 Transactions

We now take a look at the structure of transactions, as well as the different kinds of transactions:

```
pub struct Transaction {
    pub kind: TxKind,
    pub inputs: Vec<CoinID>,
    pub outputs: Vec<CoinData>,
    pub fee: u64,
    pub covenants: Vec<Melodeon::Covenant>,
    pub data: Vec<u8>,
    pub sigs: Vec<Vec<u8>>,
}

pub enum TxKind {
    Normal = 0x00,
    Stake = 0x10,

    DoscMint = 0x50,
}
```

`Transaction`'s fields have the following meanings:

- `kind`: a byte denoting what kind of transaction the transaction is. Most transactions are of type `Normal`.

- `inputs`: a vector of coins, identified by `CoinID`, that the transaction is spending
- `outputs`: a vector of coins, described by their `CoinData`, that the transaction is creating
- `fee`: total fees paid, in μmel
- `covenants`: a vector of Melodeon covenants, used to "fill in" the covenant hashes in the input coins' associated data
- `data`: arbitrary associated data
- `sigs`: a vector of "signatures", or *malleable* associated data. This is used when computing hashes: given a transaction T, the hash $H(T)$ is actually $H(T')$, where T' has an empty `sigs` field.

Applying a transaction to the world state

Now that we know the elements of each transaction T, we can describe the transaction-level state transition function Υ. Applying a transaction to the state involves three steps: Υ^I, where the inputs of the transaction are spent, Υ^O, where the outputs of the transaction are added to the state, and Υ^S, where effects and constraints of non-ordinary kinds are applied.

The algorithm of applying and verifying a transaction against a world state is described as follows. We will discuss "special' ' transactions, the fee economy, and Melodeon constraints separately.

Spending a transaction's inputs We check the Melodeon covenants of each coin, and make sure the input and output coins are balanced for each denomination. The Melodeon covenants are not stored in the coin — only their hashes are — so the transaction must supply the contents of the covenants in the `covenants` field.

Note that any of these checks may fail. In that case, the entire transaction is considered invalid and reverted, with no effect to the state.

We also note that given a set of transactions within the same block that can all be validly applied to the state in *some* order, then all other valid orders must result in the same end state, even though some orders may be invalid (e.g. if one of the transactions spends a coin created by another one). In this sense, applying multiple transactions within the same block can be considered as applying

an ordered set of transactions, despite the fact that we formulate this function as applying a single transaction. [6]

- $\Upsilon^I(\sigma, T)$:
  - **for** each `coinid` in T.`inputs`
    - **if** `coinid.txhash` is a key in σ.`stakes`, then the coin is frozen and we **abort**.
    - **remove** `coinid` $\Rightarrow$ `coindataheight` from σ.`coins`
    - **find** `cov` $\in T$.`covenants` where $H($`cov`$) =$ `coindataheight.covhash`
    - **check** that T satisfies the Melodeon covenant `cov`
  - **check** that T's inputs and outputs are balanced: for every denomination that is not the empty string, total number created (including fees) must equal total number spent. One exception: for `DoscMint` transactions, DOSC-denominated "balancing" is ignored and deferred to Υ^*.
  - **apply fees**:
    - **check** that T.`fees` $>$ Weight(T)[7] $\times$ σ.`fee_multiplier`
    - **increment** σ.`tips` by T.`fees` $-$ Weight$(T) \times \sigma$.`fee_multiplier`
    - **increment** σ.`fee_pool` by Weight$(T) \times \sigma$.`fee_multiplier`
  - **return** the changed σ

Applying a transaction's outputs No checking is done in this phase; we simply add the outputs into the state. For coins with the denomination of an empty string, we replace the denomination with the hash of the transaction; this is how Themelio implements custom tokens.

- $\Upsilon^O(\sigma, T)$:
  - **for** each ith `coindata` in T.`outputs`
    - **if** `coindata.denom` has length zero,
      - **set** `coindata.denom` to $H(T)$
    - **insert** the bincode [cra21] encoding `CoinID`$\{$`txhash` : `H(T)`, `index` : `i`$\}$ into σ.`coins`
  - **return** the changed σ

[6] It is, in fact, possible to describe an algoritihm analogous to the one in the "canonical ordering" proposal [Ver+18] for Bitcoin Cash to directly apply any valid set of transactions in linear time without actually searching for any particular valid order. But we describe the ordered, single-transaction function for clarity here.

[7] The "weight" of a transaction proportional to its execution cost, determined by its size and the opcodes used in the covenants that must run to validate it. Analogous to the concept of "gas cost" in Ethereum. [Woo14]

Applying "special" actions Here we handle all the special actions of non-`Normal` transactions.

- $\Upsilon^S(\sigma, T)$:
  - **if** $T.\texttt{kind} = \texttt{DoscMint}$
    - **let** $\texttt{cdh} = \sigma.\texttt{coins}[T.\texttt{inputs}[0]]$
    - **if** $\sigma.\texttt{height} - \texttt{cdh.height} < 100$ then **abort** (cannot measure such small time-frames accurately)
    - **let** $\chi = H_{\sigma.\texttt{history}[\texttt{cdh.height}]}(T.\texttt{inputs}[0])$
    - **let** $(d, \pi) = T.\texttt{data}$
    - **verify** MelPoW proof with seed χ, difficulty exponent d, and proof π
    - **measure** speed of minter `my_speed` as $2^d/(\sigma.\texttt{height} - \texttt{cdh.height})$
    - **let** $\delta = \frac{2^d \times \texttt{my\_speed}}{\sigma.\texttt{dosc\_speed}}$
    - **ensure** that the total output DOSC does not exceed δ
  - **else if** $T.\texttt{kind} = \texttt{Stake}$
    - **check** that $T.\texttt{data}$ deserializes to a valid `StakeDoc`
    - **check** that the first input of T is s Sym, where s is the value in `StakeDocs`
    - **check** that `e_post_end` is greater than `e_start` in the `StakeDoc`, and that `e_start` is in the future
    - **insert** $H(T) \Rightarrow T.\texttt{data}$ into $\sigma.\texttt{stakes}$.

3.5.7 Blocks

We now talk about blocks, and Ω, the "state-sealing" function that builds blocks.

State sealing

`SealedState`, representing a canonical state at a certain block height, simply wraps a `State` and an optional `ProposerAction`:

```
pub struct SealedState {
    pub state: State,
    pub action: Option<ProposerAction>
```

```
}

pub struct ProposerAction {
    pub fee_multiplier_delta: i8,
    pub reward_covhash: HashVal,
}
```

`SealedState` is produced by the state-sealing function $\Omega(\sigma, P)$. The "proposer action" P is an optional non-transaction action that the block proposer uses to pay himself the fees incurred in the block, as well as up/downvoting the fee level. More specifically,

- `fee_multiplier_delta` represents how much to change σ.`fee_multiplier`, where -128 represents the biggest possible decrease and 127 represents the biggest possible increase. The maximum amount the fee multiplier can change is $1/128$ of the fee multiplier.
  - **Note**: TIP-901 is the canonical algorithm for applying changes to fee multipliers. In particular, handling rounding when the fee multiplier is very small is rather tricky.
- `reward_dest` is the covenant hash that the fees of the block will be sent to. This is generally an address that the block proposer controls. Every block height, the proposer withdraws 2^{-16} times the fee pool, as well as all the tips. This is implemented by adding a coin "out of nowhere" into σ.`coins`.

The state-sealing function is thus:

- $\Sigma = \Omega(\sigma, P)$:
  - **apply** all Melmint/Melswap-related sealing functionality, described in its specification
  - **let** `max_movement` $= \sigma$.`fee_multiplier`$/128$
  - **let** `scaled_movement` $=$ `max_movement` $\times P$.`fee_multiplier_delta`$/128$
  - **increment/decrement** σ.`fee_multiplier` by `scaled_movement`
  - **let** `base_fees` $= \sigma$.`fee_pool`$/2^{16}$
  - **let** `total_reward` $=$ `base_fees` $+ \sigma$.`tips`
  - **set** σ.`tips` $= 0$
  - **let** `pseudocoin_id` $=$ RewardPseudo(σ.`height`)

- **let** `pseudocoin_data` be a `CoinDataHeight` with a `CoinData` that transfers `total_reward` μmel to P.`reward_covhash`
- **insert** `pseudocoin_id` ⇒ `pseudocoin_data` into σ.`coins`
- **return** `SealedState` with σ and P

State advancement

Given a confirmed `SealedState` at height n, how does the blockchain proceed? It needs to somehow "advance" the `SealedState` to a `State` at height $n + 1$. This is the purpose of the **state-advance function**, $\sigma^0_{n+1} = \Delta(\Sigma_n)$:

- Insert the current block header into `State::history`
- Advance height by 1
- Remove all stale entries in `State::stakes`
- Update `State::dosc_speed`

"Block" representation

We can now present `Block`: all the information required to get from a `SealedState` at height n to a `SealedState` at height $n + 1$:

```
pub struct Block {
    pub header: Header,
    pub transactions: im::HashSet<Transaction>,
    pub proposer_action: Option<ProposerAction>,
}
```

Note that this implies `Block` is *entirely a derived concept* for ease of serialization. So at the state-transition-function level, the Themelio blockchain does not really have "blocks" at all. "Blocks" are not a data structure that appears in the state transition function, whose serialized form is committed to by the header, or gets passed to covenants. What the canonical history of the blockchain consists of is what the headers commit to — "snapshots" of the global state, with blocks being "diffs" that allow nodes to transition the state from that committed to by one header to that committed to by the next.

Chapter 4

Consensus, trust, and transference

4.1 Introduction

Up to this point, we described Themelio as if it were a magic oracle that always correctly maintains the world state and applies updates to it for every block. In the real world, though, we require a *consensus* procedure to ensure both *consistency* — all Themelio users observe the same world state σ at every block height — and *validity* — σ can only change through valid transactions. In fact, consensus is what drives transferable endogenous trust. All security properties of blockchains are ultimately anchored in the design of the consensus mechanism.

Unfortunately, consensus is also a major source of instability and complexity in blockchains. Contentious blockchain governance problems are often tied to consensus-related issues such as decentralization (for example, the controversy surrounding Bitcoin block sizes), and attacks on existing blockchains, such as the 51% attack on Namecoin, selfish mining attacks on Bitcoin, etc., focus on exploiting problems in consensus.

Blockchain consensus also differs from consensus in other distributed systems in how it must rigorously model trust within a game-theoretical model, rather than simply making assumptions about fault tolerance. To truly achieve incentive-compatible endogenous trust, we cannot rely on the typical approach of considering ideal "honest" behavior and then positing an "adversary" with certain powers. Yet cryptoeconomic mechanism design is still a young field full of uncertainties — game-theoretical models often give results different from actual empirical observations — and creating a consensus mechanism that is incentive-compatible under a wide variety of real-world conditions has proven to be perniciously difficult. Nevertheless, we believe that we must abandon an ad-hoc approach to cryptoeconomic design — that is, designing a consensus mechanism either heuristically

or under simplified "distributed systems" security models, while analyzing the economics and incentives later, if at all.

Themelio takes a unique, "economics-first" approach to designing consensus and related cryptoeconomic mechanisms. We start by introducing a very pessimistic model: a blockchain totally controlled by one rational profit-maximizing entity. We consider what sort of blockchain rules would such a centralized business enforce, in the absence of external attackers. Perhaps surprisingly, we find that a rational monopoly would in fact behave in a trustworthy manner — the problem with centralized systems is actually "irrationality", or rather hard-to-model utility functions.

Finally, we construct Themelio's decentralized consensus mechanism. We show that a variant of proof-of-stake with a few critical departures from existing designs elegantly incentivizes a large, permissionless group of stakeholders to simulate this ideal rational entity as a whole, regardless of whether they coordinate their actions or not. This approach ensures that not only will consensus produce the desired results in a conventional "non-coordinating majority" world, Themelio's mechanism is actually *collusion-proof*: a rational colluding majority will simply act the same way as our desired rational monopoly! Thus, Themelio avoids the pervasive yet fragile reliance on coordination costs to protect blockchain security. We also design a mechanism for constructing concise *consensus proofs* that prove that the blockchain reached a certain consensus on the blockchain state, which is important for trust transference to light clients that do not run the consensus algorithm themselves.

4.2 Goals and premises

4.2.1 The problem of coordination

A commonly referred-to concept in blockchains is *coordination*. More specifically, we often see *uncoordinated majority* assumptions, where the security of a blockchain relies on assuming that no more than some small fraction (usually between 25% and 50%) of consensus participants can coordinate their actions, and that participants are rational. For example, in many major analyses of Bitcoin security [SSZ16] we assume that no more than 50% of the mining power can collude, or otherwise double-spending and similar attacks become profitable to the miner cartel.

Intuitively, this assumption seems very attractive. Coordination seems difficult in a permissionless blockchain: getting a majority of pseudonymous participants to agree to do something bad appears to be a costly "cat herding" exercise. Furthermore, collusion of a majority appears to be a scenario where the blockchain has already failed — after all, decentralization is a central pillar of the entire appeal of blockchains.

Unfortunately, we believe that this entire paradigm is problematic. This is because "coordination" is a very slippery concept. For one, to even become consensus participants, users must in a broad sense coordinate to know about a blockchain, run compatible software, etc. Already, reality contradicts a theory of narrowly self-interested parties with no knowledge of or ability to contract with each other. Moreover, consensus participants in existing blockchains clearly coordinate in many ways not anticipated by the protocol. Ethereum miners, for example, almost unanimously vote for the same block size limit [But19c], while changing their votes in unison. These changes typically happen through out-of-band initiatives in forums and such.

So what prevents consensus participants from coordinating for "evil", if they routinely coordinate anyway? Indeed, some blockchains have fallen victim to protocol-violating collusion, such as when EOS's consensus participants decided to collude in censoring a set of accounts [Flo18]. It is not at all clear what game-theoretical incentives protect other blockchains from similar attacks.

We conclude that although very commonly used in blockchain design, uncoordination assumptions do not reflect reality and should not be used in Themelio. Instead, we start with a model that assumes the opposite — a perfectly coordinated monopoly — and then use cryptoeconomic incentives to guide this model to a good outcome.

4.2.2 A "despotic" blockchain

Let us consider a "blockchain" operated by a single, profit-maximizing entity with control over the entire consensus. This entity is essentially a centralized business offering to add transactions to the blockchain for a price. What would be a rational course of action for this "blockchain despot"? We show that such a despot would, in fact, behave in a reasonably desirable manner.

A seemingly obvious objection would be that the despot would greatly abuse the users of the blockchain, through double-spending, censorship, and similar attacks. Is the whole point of decentralization not to avoid monopolies exploiting their customers? However, a *perfectly rational* blockchain operator, constrained to offer blockchain services rather than simply choose a different line of business, would not choose such a course of action. The blockchain operator would like to establish a reputation to follow protocol rules correctly, or otherwise its customers will not continue using the blockchain and its only source of revenue disappears.

More generally, we would expect the following two behaviors to emerge:

- The despot offers to include each transaction in the blockchain for a *revenue-maximizing fee*. This is the fee level at which further increases in fees would decrease revenue due to decreasing demand, and can be modeled using standard microeconomic monopoly models.

- The *minimum bribe* needed for the despot to engage in "destructive" behavior (such as disobeying the blockchain protocol) is the present value of the future lost fees of such behavior. This bribe need not be a literal bribe, but rather any sort of external payoff from disobeying the protocol. For example, if the operator also controls competing systems such as traditional payment networks, it might be in its interest to "irrationally" destroy the blockchain even without third-party influence.

Let us now show that first, the revenue-maximizing fee will not be extortionately high, and second, the minimum bribe will be a large fraction of the entire economic value generated by the protocol. This ensures that the "average" despot would be very trustworthy indeed.

Revenue-maximizing fees

The rational behavior of a revenue-maximizing monopoly is an elementary subject in microeconomics. The price that maximizes revenue for a monopoly depends on *demand elasticity* — the ratio $-\frac{\Delta Q/Q}{\Delta P/P}$ between how much the quantity demanded changes ($\Delta Q/Q$) and how much the price changes ($\Delta P/P$). For example, a good with a demand elasticity of 1 will have demand increase by 1% for every increase in supply by 1%. Generally, demand elasticity varies for the same good at different prices, increasing as price increases. To maximize revenue, a monopoly will charge at the price that gives a demand elasticity of 1, where any increase in price will decrease quantity enough to reduce total revenue.

Unfortunately, studies of demand elasticity in blockchains are few and far between. In fact, to our knowledge the best discussion of demand elasticity in blockchains is in fact a forum post by Vitalik Buterin [But18b]. It examines a few "natural experiments" where the throughput of Bitcoin and Ethereum were reduced due to unexpected network conditions, coming to the conclusion that the demand elasticity of the Bitcoin fee market is between 0.4 and 0.7, while that of the Ethereum fee market is between 1 and 2. The only published work we could find [JMU22] fits a mathematical model against Bitcoin and Ethereum data to estimate demand elasticity. But no attempt is made to limit the data to *unexpected* changes in supply, seriously weakening the conclusion they make (which includes an implausibly *negative* elasticity for Bitcoin block space demand).

Even from a ballpark estimate like Vitalik's, we already see that the fees that a blockchain despot would charge cannot be far from those charged by existing blockchain protocols. In fact, though Buterin implies that the different elasticities are due to the different applications of the two blockchains, we think that a better explanation is simply that Bitcoin's low block size limit forces it to operate at a point in the demand curve with low elasticity, while Ethereum allows miners to vote on the gas limit to maximize revenue. A rational despot would certainly charge significantly lower

fees, and produce more revenue, than Bitcoin does on average, and perhaps would charge similar fees to that of Ethereum.

The price of bribing a despot

We have established that a hypothetical despot would receive roughly the same level of fee revenue as Bitcoin and Ethereum do each year — $1\text{-}3\%$ of the yearly on-chain economic activity Y. What would be the cost of bribing this despot to destroy the blockchain? This can be easily answered by calculating the present value of future fee revenue, which the despot must give up as a result of destroying the blockchain. This would range from $0.2Y$ assuming a high discount rate of 5% and low revenue of $0.01Y$, to as much as $3Y$ with a discount rate of 1% and an annual revenue of $0.03Y$.

To put this in perspective, for Bitcoin $Y \approx 300000\text{BTC}$, or around 2 billion US dollars. Even a low-end estimate of $0.2Y$ would exceed the cost of a 51% attack on Bitcoin by many orders of magnitude. Thus, we see that *assuming the despot is rational*, bribing him to damage his own cash cow would be much harder than attacking current blockchains.

4.2.3 Decentralizing the despot

Now that we have shown that a centralized blockchain operator would, rationally, behave quite ideally. Why do we even bother with decentralization? The answer is that there are two assumptions in the analysis that we cannot rely on in the real world — that the despot is perfectly rational, and that it has a utility function valuing only monetary revenue. In the real world, centralized businesses often make irrational, non-profit-maximizing decisions, and they are subject to a wide variety of utility-function-distorting pressures such as government regulation and public opinion. Thus, we simply cannot trust that a centralized operator would in fact behave like our hypothetical blockchain despot.

However, we believe that our ideal despot is indeed a good model for a *coordinated coalition* of decentralized parties. Decentralization ensures that the utility function of the group as a whole will be that of an "average" despot, which will not have idiosyncratic factors that prevent our analysis from working. This gives us Themelio's main mechanism design goal — creating a mechanism that *simulates a rational blockchain despot* by coordinating many parties, some of which might not be able to coordinate without help.

This may seem to be a rather weird framework. After all, a blockchain despot will rationally make quite a few suboptimal choices, such as demanding fees higher than those that will maximize social utility. The reasons why we choose this approach are twofold:

- *Robustness against out-of-band collusion*: Essentially, Themelio's protocol mechanisms are designed to coordinate participants to act the way they would if they could perfectly collude out of band. Thus, we eliminate the risk of cartels breaking protocol guarantees.
- *Creates a single Nash equilibrium*: Even if we harden a traditional blockchain protocol based on a non-coordination equilibrium against malicious cartels, it would be difficult to avoid different behavior in a non-coordinated and coordinated world (say, significantly different fee levels). Such a blockchain would have multiple stable equilibria, causing uncertainty as to which "world" applications will operate under in the future and turmoil during transitions between equilibria. Themelio's approach, on the other hand, ensures stable behavior even as coalitions potentially form and disband.

We describe how we construct this decentralized despot in two parts. First, we outline Themelio's proof of stake mechanism called Synkletos [1], which uses a unique dual-token system to incentivize blockchain-despot-like behavior and mitigates the well-known "weak subjectivity" problem of proof-of-stake systems by dividing time into "stake epochs". We then look at Themelio's novel fee and reward system that supports the incentives of Synkletos through a mechanism that safely funds almost all protocol rewards through user fees.

4.3 Synkletos: consensus in Themelio

Now let us look at Synkletos, Themelio's proof-of-stake consensus mechanism. We first discuss Synkletos' tripartite division of users into stakeholders, auditors, and clients. We then examine the roles each of these classes plays in maintaining Themelio's decentralized "despot simulation".

4.3.1 Three kinds of nodes

Participants in Themelio are divided into three categories by their roles:

- *Stakeholder nodes* record transactions into new blocks and confirm them using a Byzantine fault-tolerant consensus algorithm between themselves. They communicate with each other through a broadcast protocol which other nodes in the network never participate in. Anybody can become a stakeholder by locking up a significant amount of cryptoasset as collateral, similar to other PoS systems like Casper. Stakeholders correspond to miners or validators in other systems.

[1] From σύγκλητος, Greek for "senate", a reference to the Byzantine Senate

- *Auditor nodes* download newly created blocks from the stakeholders and gossip them between themselves while mirroring the entire world state. Anybody can join the network as an auditor by simply running a piece of software. Auditors verify new blocks decided by the stakeholders and check that the stakeholders never equivocate on the content of a given block height. Auditors roughly correspond to full nodes in other systems, although they have a more important role in Themelio's security.
- *Client nodes* are lightweight participants that query the network of auditors to access specific information in the blockchain, yet do not trust any particular auditor.

Essentially, the stakeholders create blocks through a proof-of-stake consensus, but auditors check their results, and everything can be queried by clients. Let us now see how this actually works.

4.3.2 Stakeholders: the oligarchy

Themelio uses a variation on a classic technique known as bonded proof of stake, used in systems like Tendermint and Casper. We keep track of a special secondary currency on the blockchain known as the *sym*. Syms are traded freely alongside mels, the main cryptocurrency of Themelio, with a relatively unchanging supply. They can be thought of as equity "shares" in the distributed blockchain despot, sharing both control and revenue.

In order to become a stakeholder, one *stakes* a sum of syms, locking them up for a fixed period of time (at least 500,000 blocks, or approximately 6 months) as a performance bond. During that period of time, the stakeholder obtains voting rights in a consensus algorithm in proportion to the amount of syms staked. Furthermore, the stakeholder staking x% of all staked syms receives revenue in proportion to x% of all economic value provided by Themelio. This ensures that the value of a sym v is proportional to the total economic value of Themelio Y divided by the number of staked syms S: $v \propto Y/S$. We will discuss how this is accomplished in Section 4.4.

Byzantine fault-tolerant consensus

The stakeholders come to agreement on each block through a partially synchronous Byzantine fault-tolerant (BFT) consensus algorithm. Formally, we abstract this as a protocol $(\sigma', \pi_{\sigma'}, B) := \mathsf{BFT}(\sigma, T^*_{\text{tentative}})$ deciding the next block from the current state, run at every stakeholder node, where

- σ is the current world state

- $T^*_{\text{tentative}}$ is a tentative proposal of transactions to include in the next block, which may not be consistent across all stakeholders.
- B is the newly produced block, consistent across all honest stakeholders.
- $\sigma' = \Omega(B, \Upsilon^*(\sigma, T^*_{\text{final}}))$ is the new block state, consistent across all honest stakeholders, where T^*_{final} is the decided set of transactions in the new block.
- $\pi_{\sigma'}$ is a *consensus proof* consisting of a list of cryptographic signatures by stakeholders, each signing $\mathsf{B2H}(\sigma')$.

BFT also has the following three properties:

- *Accountable safety*: as long as at least $2/3$ of the syms staked by the stakeholders belong to protocol-following stakeholders ("there is a quorum"), all honest stakeholders agree on the same σ', and σ' must be the result of applying a valid block-level state transition to σ. Furthermore, every protocol-following stakeholder will only sign one proposal for σ', so two valid consensus proofs of the same block height will never occur if there is a quorum.
- *Liveness*: as long as there is a quorum, the protocol will make progress.
- *Fairness*: each stakeholder's proposal is accepted roughly in proportion to its share of all staked syms.

In the current implementation of Themelio, we use as an implementation of BFT Streamlet [CS20], a Byzantine fault-tolerant consensus algorithm designed for simplicity and obvious correctness. Interestingly, though, the exact choice is not really important This is because the consensus proof π'_σ encapsulates the result of the consensus algorithm, and the rest of the details are invisible from the rest of the network. This means that the BFT consensus algorithm itself is actually not part of the core state-transition function of the blockchain — the "consensus-critical" core that must stay stable over time! Any changes must be backward-compatible, but fundamentally as long as compatible consensus proofs are generated, stakeholders are free to switch to different BFT consensus algorithms. The incentives, and not the exact implementation, of Themelio's consensus algorithm is the portion crucial to Themelio's endogenous trust.

Why proof-of-stake?

Why did we choose PoS over other consensus algorithms, such as the venerable proof of work (PoW) of Bitcoin, or the proof of authority (PoA) found in consortium and private chains? There are both

general reasons that apply to all proof-of-stake systems, and also a reason more specific to how Themelio uses proof of stake. The Ethereum Proof-of-Stake FAQ [But19b] gives a strong defense of PoS; some properties of PoS in general we consider especially important for Themelio include:

- *Higher security margin*: Attacking a PoS blockchain directly requires expending a vast amount of resources to buy up stake. This is equivalent to around 1/3 of the total value of staking coins (a proxy of the economic value of the blockchain system). Thus, as usage increases, PoS security will proportionately strengthen until it becomes practically invulnerable to attacks on the consensus protocol. Proof-of-work blockchains like Bitcoin, however, can be subverted quite cheaply. Attacks reverting a full hour of Bitcoin transactions cost less than $1,000,000 [cry19], pocket change compared to the almost $100 billion Bitcoin market capitalization. Finally, proof of authority, which is not a decentralized solution, is very fragile to centralized attack vectors such as hacking or government regulation.
- *Immediate finality*: PoS allows easy, secure finality using asynchronous Byzantine fault-tolerant consensus protocols. This means that as long as we have a quorum of stake owned by correctly behaving nodes, a block that is successfully appended to the blockchain will never be reverted. This eliminates the unpredictable behavior found in "chain-based" consensus protocols like proof of work, such as forks and block reorganizations. More importantly, it eliminates the implicit trust in network synchronicity found in "heaviest-chain" style consensus algorithms, preventing eclipse and similar attacks.
- *Stronger incentive compatibility*: As we will see shortly, staked bonds allow us to punish misbehaving stakeholders by deleting their stake. In other blockchains, misbehaving miners only lose potential rewards or reputation. As the Ethereum FAQs put it, "in PoW, we are working directly with the laws of physics. In PoS, we are able to design the protocol in such a way that it has the precise properties that we want - in short, we can optimize the laws of physics in our favor."

A more important reason why we use proof of stake with a special staking token, however, is that we conjecture that it *distributes the utility function of a hypothetical despot* to all participants. Intuitively, this is because the stakeholders as a whole receive the same consequences of their collective actions as a despot, while syms, being valued as fractions of the revenue of the stakeholder collective due to BFT's fairness property, would depreciate and appreciate to replicate these consequences in proportion to the number of syms someone owns. This relies crucially on the fact that the sym is an "equity" share; a single proof-of-stake system using a currency-like coin, such as that used by Ethereum, would not have such an elegant incentive structure. We test this claim empirically in Section 4.5.

A more complete account of how the incentives work can be found in Section 4.4.

Performance vs decentralization

Scalable blockchains with immediate finality cannot support an unlimited number of consensus participants. This is because Byzantine fault-tolerant consensus algorithms have rapidly increasing overhead with increasing participants. In practice, performance constraints probably limit the number of consensus participants to a few hundred.

However, we intentionally do not elect these participants from a larger pool using "coin-voting" — a common method of deriving a small amount of participants from a large body of coinholders called *delegated proof of stake* (DPoS). In DPoS, coinholders vote for people with voting power proportional to their coin ownership, and only the few with the most votes become "delegates" and participate in consensus. EOS [EOS22] is a popular blockchain using DPoS.

Yet although DPoS gives a vote to all coinholders, it insulates coinholders from protocol incentives. Coinholders are not responsible for the actions of the delegates they vote for, while misbehaving delegates receive no punishment other than a loss of reputation. Thus, coinholders have no incentive to vote for "good" nodes, delegates have little incentive to behave correctly, and misbehavior is rampant. Unsurprisingly, all the problems of political governance in a representative democracy get imported. Elections involve massive advertising campaigns, vote-buying, and even nationalist agitation [Blo18], while delegates often behave as a centralized cartel, engaging in actions like censoring transactions [Flo18].

Sortition is another approach, used most notably in Algorand [Gil+17]. Periodically, a committee of participants is randomly selected from all coinholders — each coinholder has a probability to win this "lottery" in proportion to the coins that they hold. The committee then participates in a consensus protocol to decide new blocks until the next lottery comes around.

Sortition eliminates most of the politics-like problems of DPoS, allowing protocol incentives like rewards and slashing to work fairly well. Unfortunately, important problems remain. Randomly selecting participants trustlessly turns out to be a rather hard cryptographic problem — a corrupt lottery can reliably elect malicious committees, and solutions like Algorand's use [Gil+17] of verifiable random functions require compromises in stricter liveness assumptions and greater complexity. Bribery attacks also become much easier, since instead of buying 1/3 of the coins, attackers can simply bribe the current committee, who has only a small fraction of the stake. Complex consensus protocols and advanced, non-quantum-resistant cryptographic techniques can reduce both challenges. But "fancy" mechanisms generally go against Themelio's philosophy of future-proof simplicity.

A point must be made that *blockchain consensus is not analogous to political governance*. Themelio's "plutocratic oligarchy" of stakeholders certainly does not make for an effective way of electing a parliament. But for blockchains, it yields highly robust and decentralized security, because we do

not pursue decentralization for its own sake. Instead, it is simply to average out individual interests to simulate a blockchain despot, implying that *decentralization has rapidly diminishing returns*.

Thus, we do not believe that Themelio's "plutocratic" bonded proof of stake is any more vulnerable to centralized threats than PoS blockchains that attempt to maximize the number of participants. Even so, as we will immediately see, Themelio has a system of *auditors* keeping stakeholders in check, ensuring that even a fully corrupted quorum of stakeholders cannot do much damage.

4.3.3 Auditors: keeping stakeholders in check

Making failure catastrophic

The second role in Themelio belongs to the *auditors*. Auditors are "full nodes" in usual terminology, replicating and validating the entire blockchain. They form a random *gossip* network among themselves, similar to that used by Bitcoin full nodes. Through this gossip network, information about new blocks is disseminated. Gossip reduces load on the stakeholders and makes it difficult for malicious networks to censor the blockchain — as long as some auditors can connect to the stakeholders and the auditors form a connected graph, new blocks will eventually be visible to every auditor.

The more important role of auditors, though, is to *make consensus failure catastrophic*. This plays a crucial role in replicating the utility function of a blockchain despot to the oligarchy of stakeholders. Auditors utilize their position as relayers of new blocks to continually monitor for cryptographic evidence that the stakeholder consensus is corrupt. This is either a consensus proof π_σ with an invalid state σ ("invalid state"), or two consensus proofs $(\pi_{\sigma_1}, \pi_{\sigma_2})$ where $\sigma_1.\mathsf{height} = \sigma_2.\mathsf{height}$ but $\sigma_1 \neq \sigma_2$ ("equivocation"); we call such a message a *consensus nuke*.

Any auditor that sees a consensus nuke immediately broadcasts it to all auditors it knows in the gossip network. It then permanently activates a "kill switch" and refuses to operate normally. Thus, an attempt at forking or appending invalid transactions to the blockchain would destroy the entire network.

Why consensus nukes?

This objective seems a little strange. Why would we ever want our network to self-destruct?

The obvious answer is that if we no longer have a 2/3 supermajority of honest stake, the entire system is irrecoverable. More specifically, a well-known result [DLS88] mathematically proves that consensus protocols running in a partially synchronous network model (that is, network delays are

unknown but finite) cannot possibly tolerate more than 1/3 arbitrary faults. So we have to choose between a model where the network stays up, but malicious stakeholders can corrupt the state arbitrarily (rewriting history, giving themselves free money — or shutting down the network), or one where the only thing a corrupted quorum can do is shut down the network. Clearly, the latter is preferable.

Much more importantly, consensus nuking changes the incentives of potential attackers by making breaking the consistency of the blockchain unprofitable for our hypothetical blockchain despot. Consider a blockchain where consensus-breaking attacks (like Bitcoin's 51% attack) allow arbitrary state corruption. A malicious actor with control over consensus can extract huge profits simply through double-spending. With more complex higher-level applications relying on blockchain data, profit opportunities are even more numerous. Thus, if self-serving stakeholders rationally collude, they are greatly incentivized to attack the network and destroy its security guarantees.

If a successful attack can only result in the network stopping all work, an "average" profit-maximizing despot will not execute such an attack and destroy their future revenue stream. Instead, only attackers who benefit from destroying the network will participate, and even then, they must control a vast amount of syms to be able to launch a consensus-breaking attack.

Finally, a shutdown when a successful attack occurs forces Themelio users to manually coordinate an emergency "hard fork" out-of-band to restore the network — a loss of endogenous trust, but a safer failure mode than the attackers successfully corrupting state. This would involve, at the very least, a redistribution of stakes away from the attacking parties and possibly protocol improvements to prevent future attacks. On the other hand, if the blockchain continues to operate even when stakeholders are corrupting the state, nothing forces users to coordinate a hard fork. It is conceivable that the malicious stakeholder cartel can create a climate of pressure for users to go along with the corrupted chain — for example, the state corruption might be forced by legal regulation or presented as way of restoring stolen assets. Consensus nuking ensures that these scenarios are impossible.

An important caveat is that consensus nukes alone are insufficient to stop *censorship* attacks, where stakers collude to *not* include certain transactions in the blockchain, perhaps due to external pressure, since they do not leave cryptographic proof that can be gossipped. Nevertheless, as long as the auditor network is functional to the extent that messages from honest auditors eventually reach other honest auditors, telltale signs of censorship will quickly propagate, such as valid transactions being repeatedly gossipped without being added to a block. It is probably possible to devise a protocol to degrade the network to damage staker income when this happens, but currently we simply assume that the reputational damage this will do to the network is a sufficient deterrent.

4.3.4 Clients: scalable consumers of endogenous trust

Most users of a blockchain, Themelio not excepted, do not have nearly enough resources to process all transactions 24/7. Users that do not synchronize the whole blockchain state, known as light clients, serve a vital role in any blockchain system. In other blockchains, though, light clients come with both reduced security and mediocre performance. Bitcoin, for example, has light clients who must persistently store a growing set of block headers and connect to at least one trusted full node.

In Themelio, light clients (usually just called clients) are thinner, safer, and much more general than light clients in commonly used blockchains like Ethereum and Bitcoin. Clients only synchronize a small piece of data, less than a kilobyte in size, a few times a year. Yet with this data, they can fully validate a large variety of information they can freely obtain from auditors. Even if a client only connects to bad auditors, it cannot be fooled into accepting invalid data. We accomplish this through two features: *state commitments* in block headers and *stake epochs*.

State commitments

As we have previously seen, at every block height h Themelio cryptographically commits to its world state σ_h through a block header. This allows auditors to generate short proofs about the content of the current world state. More specifically, given any key k and an SMT tree root (for example σ_h.`transactions`), an auditor with access to the world state can generate either a proof that k is bound to a specific value v, or that k does not exist in the mapping. This proof is only of size $\Theta(\log n)$, where n is the number of values in the SMT.

What this means is that given that it somehow obtains the latest block header from a trustworthy source, a light client can easily check the status of coins and transactions by simply asking an untrusted auditor. This allows clients to trustlessly interact with Themelio's state in quite a complex manner. Note that although this process is reminiscent of Bitcoin's SPV (simple payment verification) protocol [Nak08], a major difference exists in that Bitcoin does not commit to its world state in a way that allows easy proofs. Instead, Bitcoin only commits to the content of each block, allowing for proofs that a transaction was included in a block, but the actual status of coins in the world state can only be trustlessly determined by downloading all transactions and applying them to the genesis state. Thus, even very simple Bitcoin applications, such as wallets, cannot operate with truly endogenous trust and must rely on a trusted full node.

Of course, all this assumes that the client can in fact obtain the latest block header trustlessly. This is accomplished by the second pillar of Themelio's light client system, stake epochs.

Stake epochs

How are clients supposed to get the latest block header? In many blockchains, clients simply synchronize *all* the block headers. Clients can then use the consensus proof embedded in each header to verify the next. In "heaviest-chain" proof-of-work blockchains such as Bitcoin, this consensus proof is simply a valid proof-of-work solution. In Themelio's case, it would be the value π_σ produced by BFT.

Unfortunately, such a strategy would be prohibitively expensive in Themelio. π_σ contains cryptographic signatures from at least 2/3 of the stakeholders — this is at least 64 bytes (the size of an ed25519 signature) for every stakeholder, and would take up dozens of kilobytes. Combined with Themelio's 30-second block interval, the block header consensus proofs spanning a year would amount to more than 10 GB.

To fix this problem, Themelio uses a strategy that limits when the stakeholder set can change, a technique also found in other protocols, like the Vault [Leu+18] fast bootstrapping protocol for Algorand. We divide blocks into *epochs* lasting 500,000 blocks, or about half a year. Within each epoch, the list of stakeholders and their respective voting weights stays the same. All stake-related transactions, such as staking and slashing, take effect *only at the start of the next epoch*. When validating the hth consensus proof π_{σ_h}, we actually use the stake document $\sigma_{\lfloor (h-1)/500{,}000 \rfloor}$`.stakes` rather than the immediately preceding one.

Through this process, a client can quickly "skip back" until it reaches a stake document it already knows. For example, to validate block header 1,100,000, we need the stake document embedded at height 999,999. But to validate the world state that height, we need the stake document at height 499,999, etc.

Thus, to synchronize with the clients simply have to catch up on all the new stake documents they missed — a few dozen KB every 6 months. Afterwards, they can securely validate the latest block header, which then lets them query the content of the world state without trusting anyone. Such ultra-light clients allow very good *read scaling*: billions of small devices like smartphones can easily query the world state while keeping trust totally decentralized.

4.4 Fees and incentives

In this section, we describe how Themelio's unique fee and incentive structure coordinates the behavior of a decentralized group of stakeholders to approximate that of a "blockchain despot". This is accomplished by a fee mechanism that incentivizes stakeholders to charge fees at the same level that a despot would charge, while backing the value of their stakes with these fees.

4.4.1 Syms as equity shares

Earlier, we mentioned that Themelio uses a special secondary currency, the *sym*, in its proof of stake mechanism. We hand-waved it as having "equity-like" properties, but what exactly does that mean?

Consider the revenue-maximizing blockchain despot that has no external incentives. Such a despot would charge fees as to maximize the economic value captured from the Mel protocol. Let us assume that Themelio's protocol can somehow charge fees exactly the way a despot would, and that these fees are distributed to stakeholders in proportion to their stake. For example, if there are only two stakeholders, Alice staking 10 syms and Bob staking 30 syms, and the despot-simulating protocol collects 100 mels of fees a day, Alice would be paid 25 mels/day while Bob would be paid 75 mels/day.

In such a model, staked mels are simply shares in the revenues of this simulated despot. Assuming that markets are efficient, then, the real value of each stakeholder's stake is proportional to that of the discounted future revenue of the stakeholders as a whole. This is crucial to our goal of having the stakeholders coordinate to simulate a single entity — any action that would harm or benefit a despot would have proportionate harms or benefits to every single stakeholder.

For example, censoring transactions, an action that generally requires majority coordination, is likely to reduce the long-term economic value of Themelio and thus the revenue stream of our hypothetical despot. Thus, a rational despot with no external incentives will not choose to censor transactions. In Themelio's actual world of disparate stakeholders, coordinating to censor transactions will similarly harm each and every stakeholder, and thus no rational stakeholder would have an incentive to participate in such a malicious act. Furthermore, an adversary wishing to bribe stakeholders to take value-harming actions will have to compensate each stakeholder for their expected loss of revenue — a value equivalent in total to bribing a despot.

There are two remaining pieces in the puzzle:

- First, *how do we generate a stream of fees that simulates the income of a revenue-maximizing despot*? The solution is a fee and reward system very different from that of existing cryptocurrencies, which we will show incentivizes even uncoordinated stakeholders to coordinate in charging fees at the right level.
- Secondly, *how to prevent failures due to discoordination*? Unlike most blockchain protocols that assume uncoordinated participants, Themelio assumes a world where stakeholders may rationally choose to coordinate their actions. Yet we must avoid prisoner's dilemma-type situations where uncoordinated rational choices produce outcomes divergent from that of a unified despot. This is done through two mechanisms in the incentive system — slashing and nuking.

4.4.2 Stable and incentive-compatible fees

As in Bitcoin and other public blockchains, each transaction in Themelio includes a transaction fee to compensate stakeholders and make flooding attacks costly. Most other blockchains let transaction senders voluntarily decide whatever fee they like; block creators then decide which transactions to include in the limited space within a block. This functions as a pretty fair and efficient first-price auction, since transactions with more fees relative to the burden they pose to the network get higher priority. Unfortunately, auction-based transaction fees paid to whoever included the transaction in a block have two significant problems:

- *Volatile, unfriendly fees*. When blocks are filled, average fees will vary quite a lot as demand fluctuates. In practice, persistently full blocks are the norm, whether due to demand increase in protocols like Bitcoin where the block size cap is fixed, or due to block producers setting block limits according to demand as in Ethereum. Thus, fees for full blocks are extremely volatile in existing blockchains, often changing as much as 2x within one block interval. This makes for a very poor user experience.

  Furthermore, it is difficult to determine how much fees to bid in order to get transactions confirmed in a traditional fee market. Wallets need complicated algorithms to estimate the right amount of fees based on looking at unconfirmed transactions — which light clients cannot even securely monitor. All this tends to greatly increase the friction of using the blockchain.

- *Coordinated and uncoordinated rationality sharply diverge*. Finally, a fee market based on a first-price auction has severe incentive problems. In the most well-known instance, if participants' have revenue largely derived from fees — a requirement for Themelio's fee-backed syms to work — pathological, protocol-violating behavior can result under an uncoordinated rationality assumption [Car+16]. Other blockchains typically mitigate the worst of the problems by funding consensus participation primarily with monetary inflation, a choice Themelio cannot make.

  The real problem underlying the game-theoretical "nastiness" of first-price fee auctions, however is that even ideal uncoordinated rational behavior is very different from coordinated rational behavior. In particular, ignoring pathological cases like selfish mining [Car+16] that involve one rational actor controlling a large minority of all consensus power, uncoordinated block creators normally want to include as many transactions as possible in their own blocks as long as they pay fees exceeding the cost to process them. This is in contrast to a "despot-simulating" coordinated cartel, which would want to reject some transactions that would be profitable on the margin in order to jack up the average fee level towards the revenue-maximizing point. As a rule of thumb, games where uncoordinated equilibria greatly differ

from coordinated equilibria tend to be prone to pathological strategies — and in any case, our design philosophy strongly favors collusion-proof incentive structures.

Thus, we abandon the traditional fee auction model in favor of a system inspired by EIP-1559 [But+14b]. Recall that as specified in Section 3.5.6, transactions have a mel-denominated `Transaction::fee` field, which denotes the transaction fee. This fee is actually the sum of two components: a mandatory *base fee* and discretionary *tip*. The base fee $\mathsf{BaseFee}(t) = \beta \cdot \mathsf{Weight}(t)$ for a transaction t is calculated by multiplying by the *base fee multiplier* β and the *weight* of a transaction, a metric that roughly measures its cost. Tips are any transaction fees Transaction senders can then add a *tip* above and beyond the base fee.

Every time a new block is created, the stakeholder proposing the block can adjust the base fee multiplier by up to 1% upwards or downwards — the base fee multiplier then reflects the stake-weighted median of the stakeholders' preferences. [2] Base fees are deposited into a special *fee pool* regardless of who included the transaction into the blockchain; the stakeholder creating a block then withdraws a tiny fraction ($1/65536$) of the fee pool. The net effect is that the base fee of a transaction is distributed to all stakeholders regardless of who made the block that contains the transaction.

Tips, on the other hand, are simply paid to the block producer, like fees in traditional blockchains. Tips will likely be a small fraction of total fees, and they give an incentive for block producers to actually include transactions instead of freeloading on a fee pool replenished by other, more honest block producers.

Figure 4.1 illustrates the flow of funds every time a new block is created.

Themelio's fee system fixes both fee unpredictability and incentive compatibility. It is comparatively easy to see why fees would be much more predictable — base fees are charged at a publicly announced, slowly-changing value, and "base fee + small tip" will be a working strategy for clients in almost all situations.

Incentive compatibility is a little trickier. We assume that there is a single revenue-maximizing fee multiplier $\hat{\beta}$, where for a blockchain despot, charging a fee of $\beta \cdot \mathsf{Weight}(t)$ for each transaction t is the revenue-maximizing strategy. Furthermore, we assume that given any two multipliers β and β' where $|\beta' - \hat{\beta}| < |\beta - \hat{\beta}|$, and both β and β' are either greater or lesser than $\hat{\beta}$, β' generates more revenue in the long run than β. Informally, this just means that nudging a suboptimal price closer to $\hat{\beta}$ would always be profitable for a despot.

Let us now analyze one specific situation — one specific stakeholder assembling a block to propose as the next block in the blockchain. In a world of perfectly coordinated stakeholders, this stakeholder will of course nudge β closer to $\hat{\beta}$ and include as many valid transactions as possible, in

[2] This is similar to how Ethereum miners vote for the maximum block size

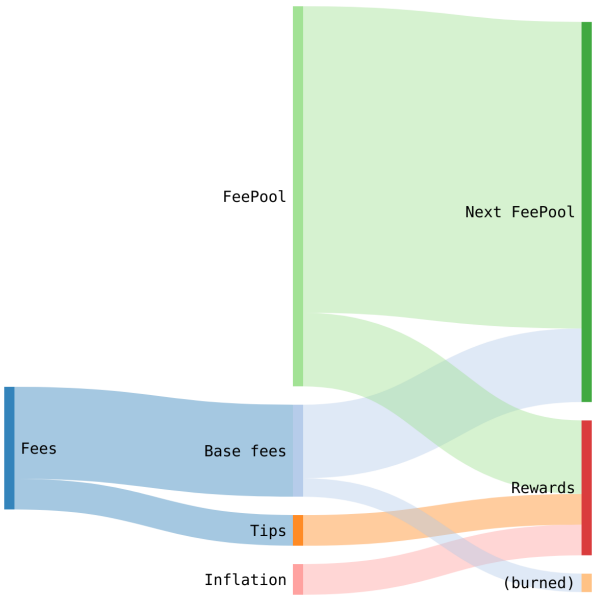


Figure 4.1: Per-block producer rewards (including sym inflation)

accordance with the strategy we assume is optimal for a despot. Furthermore, valid transactions will be included even without tips, since setting the base fee signals the reservation price of the stakeholder cartel much better than refusing to accept transactions without the right tip. Let us now show that even when the stakeholder must make a decision without coordinating with others, he will in fact behave the same way. More precisely, we show that

1. Tips constitute a negligible portion of the fees charged by the stakeholder.
2. The base fee multiplier β will be set as close to $\hat{\beta}$ as allowed by the protocol.

Showing that tips will be negligible is fairly easy. In an uncoordinated world, the stakeholder building a block is incentivized to include every transaction whose tips exceed its marginal processing cost, since otherwise some other stakeholder will include the transaction instead. Tips converge to marginal processing cost, just like how prices converge to marginal cost in any perfectly competitive market.

Empirically, this marginal cost is exceedingly small compared to the revenue-maximizing fee. For example, transaction fees in Bitcoin were extremely low before increasing demand led to block space becoming a scarce resource, averaging around a few cents per transaction, compared to the

	Single stakeholder	Despot
Increase β	$f \cdot R(\beta + \delta)$	$R(\beta + \delta)$
Decrease β	$f \cdot R(\beta - \delta)$	$R(\beta - \delta)$

Figure 4.2: Payoff table for adjusting β by δ

multiple-dollar fees after Bitcoin blocks filled up [Blo21]. Yet these minuscule fees must have been sufficient to cover the marginal cost of processing a transaction, for miners would otherwise have not included them in the blockchain.

Let us see why our second property holds: that the base fee multiplier continually approaches the revenue-maximizing level. Assuming that tips are negligible, let us consider the payoffs of a single stakeholder and a blockchain despot, in deciding whether to decrease or increase β by δ. This is shown in Table 4.2, where $R(m)$ is the present value of the future revenue generated by setting the fee multiplier to m, and $0 < f < 1$ is the fraction of all staked syms owned by the stakeholder in question.

We see that the rational decision for both a single stakeholder and a perfectly coordinated "despot" is the same — adjust β in the direction that maximizes $R(\beta)$. This is because if we assume uncoordination, moving β in the "right" direction reduces the expected future distance between β and $\hat{\beta}$, *regardless of what choices others might make*. Thus, even in a completely uncoordinated world, stakeholders have an incentive to nudge β towards our desired value $\hat{\beta}$.

What about the vast space of situations in between perfect coordination and perfect uncoordination? We can actually largely ignore it due to the following observation: stakeholders forming n internally-coordinated coalitions that do not mutually cooperate is essentially the same as n perfectly uncoordinated individual stakeholders — in both cases, we have n uncoordinated profit-maximizing agents acting.

4.4.3 Emergency responses: slashing and nuking

We have now shown that under a wide variety of scenarios, Themelio's stakeholders will rationally charge stable and predictable fees, while admitting transactions in a nondiscriminatory manner, just as a hypothetical blockchain despot would. However, we left out a *very* important job of stakeholders that's surprisingly hard to incentivize — they need to actually validate transactions and make sure that valid blocks are produced.

In a perfectly coordinated world, this is easy: producing invalid blocks that will not be accepted by auditors only serves to halt the network, which is not in the interest of most stakeholders.

	Others validate	Others do not
I validate	$r - c$	$r - c$
I do not	r	$-R$

Figure 4.3: Payoff table for choosing to validate transactions or not

Unfortunately, if we assume uncoordination, validating transactions correctly may not be a dominant strategy.

To see why, let us look at Table 4.3. r represents the reward from building a block, c the cost of checking whether or not the transactions inside a block are valid, and R the harm to the stakeholder from an invalid block being confirmed onto the network and reducing Themelio's economic activity.

We note that if the stakeholder thinks that other stakeholders do indeed validate transactions, the rational strategy is in fact to include whatever transaction the stakeholder comes across, without checking for validity. After all, if most stakeholders validate, users will have no incentive to send invalid transactions and invalid transactions will in any case not be propagated within the peer-to-peer network.

We therefore have a serious coordination failure — in a world where stakeholders all validate transactions, each of them would instead be incentivized to free-ride on the effort of others and not validate at all. We get the worst of all worlds, where nobody validates transactions and the entire system collapses. In fact, an analog of this "lazy stakeholder" strategy, SPV mining, is already common in the Bitcoin world, and the only thing preventing it from leading to consensus failure is, effectively, out-of-band coordination.

To fix this, Themelio contains a mechanism widely used in other proof-of-stake systems, though usually not motivated explicitly as a way to avoid coordination failure. This mechanism is *slashing*, where stakeholders that behave in provably invalid ways are punished by deleting their entire stake. More precisely, the last step of our consensus protocol BFT has all stakeholders *commit* to a particular block by signing it cryptographically, a process that builds the consensus proof produced by the protocol. Correctly behaving stakeholders will always commit a valid block and never "go back" on their collective decision. Thus, we can define a *proof of equivocation* which leaves cryptographic proof that a certain stakeholder is faulty: signatures on two different blocks with the same block height, from the same stakeholder.

Anybody can submit proofs of equivocation as specially-formatted transactions on the blockchain. This *slashing transaction* removes the offending stakeholder, deleting all of the syms associated with the stake. Slashing also reduces the supply of syms, increasing their value, while also increasing the fraction of rewards that other stakeholders receive. This incentivizes every other stakeholder to

discover and slash "lazy" stakeholders, perhaps to the point of intentionally sending them invalid transactions in the hopes of catching and slashing a lazy validator.

In this way, slashing turns the payoff of "others validate but I do not" deeply negative, leading to transaction validation being a dominant strategy even in a totally uncoordinated setting. We can now also understand the *consensus nuking* discussed in Section 4.3.3, where contradictory consensus proofs propagate through the auditor network and shut the entire system down, as an extraordinary form of slashing that punishes lazy or dishonest majorities.

Nuking provides two additional benefits on top of slashing. First, it is a failsafe that protects network integrity if stakeholders "irrationally" (for example, due to a software bug) fail to validate transactions. Secondly, nuking also prevents the use of a wide range of "despot-compatible" lazy strategies that are nevertheless undesirable, such as choosing to skip validating "probably okay" transactions to save costs when most clients would not attempt to send invalid transactions. Nuking makes the consequences of even slightly invalid behavior terrible, preventing these strategies.

4.5 Agent-based incentive simulation

In this section, we tentatively test Synkletos's security and stability with a series of experiments. We use a Monte Carlo *agent-based simulation* approach well-known in the literature [Chi+19]. We build a simplified model of Themelio's transaction market, including both users wishing to broadcast transactions and stakeholders building blocks out of transactions. This then lets us compare the payoffs of different stakeholder strategies under a variety of environments.

Through this simulation, we demonstrate that a range of "obvious" stakeholder strategies — we call them *standard* strategies — converge to despot-simulating behavior with minimal tips and revenue-maximizing base fees. Furthermore, we show that a variety of pathological strategies, such as lazy validation or continually voting down the fee multiplier, can cause gross deviation from despot-simulating behavior when they are in the majority, yet stakeholders following standard strategies receive higher payoffs even when they are in the minority. We do note, though, that this is intended as heuristic evidence in favor of Synkletos' stability, and not an exhaustive exploration of possible strategies.

Finally, we run simulations on variations of Synkletos, for example by burning all the base fees as in EIP-1559. We see that Synkletos' various design choices are crucial for its stability.

4.5.1 Simulation model

Our simulation consists of three parts: the *world state*, the *transactions*, and the *stakeholders*.

World state Time in our simulation is discrete, divided by block heights. The world state W_t at height t consists of the fee multiplier $W_t.\mathsf{multiplier}$, the fee pool $W_t.\mathsf{pool}$, and the transaction queue $W_t.\mathsf{txqueue}$. The transaction queue represents transactions that have been submitted to the blockchain but not yet confirmed.

Transactions Every block height, a batch of random transactions is generated and added to the world state. Each of these transactions T is represented by three numbers: a number $T.\mathsf{weight}$ proportional to the cost of validating it, its fee $T.\mathsf{fee}$, and $T.\mathsf{maxfee}$ that represents the highest fee that its sender is willing to bid. We do not attempt to model invalid transactions or the actual on-chain effects of transactions.

To generate a transaction T at time t, we first sample a *reservation fee level* r from an exponential distribution with mean 1. This represents the highest price per weight unit that the sender is willing to pay. If $r < W_t.\mathsf{multiplier}$, then even the base fees are too much for the sender to pay, and the transactions to discarded. We set the fee to 1% more than required:

$$T.\mathsf{fee} = W_t.\mathsf{multiplier} \times T.\mathsf{weight} \times 1.01$$

and we set $T.\mathsf{maxfee} = r \times T.\mathsf{weight}$.

Furthermore, every time a transaction in the world state's queue is not accepted into a block, its fee increases by a random amount between 1% and 50%. If this fee exceeds its max fee, then the transaction is deleted from the queue. This process simulates an "auction-like" process that bids up tips when the base fee is set too low.

Stakeholders Within a single simulation instance there are n stakeholders $S_1, \ldots, S_n$; stakeholders joining and leaving is not modeled. We model the costs that S_i experience with two variables:

- $S_i.\mathsf{fixcost}$ representing the fixed costs (such as server rent) of running a stakeholder for 1 block height
- $S_i.\mathsf{dyncost}$ representing the dynamic costs of processing a transaction. That is, if a stakeholder confirms a transaction T with weight w, then it must pay $w \times S_i.\mathsf{dyncost}$.

Every block height t, a random stakeholder is selected to be the block proposer. They pick transactions out of $W_t.\mathsf{pool}$ based on their *transaction picking strategy* to confirm. Confirmed transactions' tips go to the proposer, while their base fees are added to the fee pool. We name the following *standard* transaction picking strategies that generally converge towards good behavior:

- Altruistic picks transactions in the interests of all stakeholders. It accepts a transaction if and only if its total fees (its long-run benefit to all stakeholders' revenue) exceed its total cost to all stakeholders.
- Greedy picks transactions in the interests of the proposer alone. It accepts a transaction if and only if its benefit to the proposer's marginal revenue — calculated as tips plus a fraction f of the base fees corresponding to the proposer's fraction of the total stake — exceeds the cost to the proposer itself.

as well as the following *pathological* strategies:

- Lazy never picks any transaction. Proposers following this strategy hope to "free ride" on the fee pool funded by other proposers' efforts.
- Monopolist picks up all transactions, hoping to deprive others of revenue.

After the stakeholder is done picking transactions, it uses a *fee adjusting strategy* to adjust the fee multiplier by at most 1%. We only define one standard adjusting strategy:

- HillClimb randomly chooses between increasing the multiplier by 1% or decreasing it by 1%. If the current multiplier is below the multiplier at which the stakeholder has seen its highest profits, the probability of increasing is 70% while that of decreasing is 30%. Otherwise, the probability is reversed. This gradually moves the fee towards the level most profitable for the stakeholder.

4.5.2 Standard strategies

We start our experiments by testing our two standard strategies, Greedy and Altruistic.

Standard strategies converge We first show that both Greedy and Altruistic converge towards despot-simulating behavior at various levels of collusion. This is done by running our model for 1000 iterations with different numbers of stakeholders, where all stakeholders use either Greedy or Altruistic to select transactions. We also initialize all stakeholders with the same fixed cost of 1000 and dynamic cost of 1 unit.

Figure 4.4 shows the *base fee ratio*, or the fraction of fees paid as base fees rather than tips. A despot will keep this ratio close to 1, meaning that most fees are paid as base fees, since this gives the same revenue while reducing the number of times transactions have to be retried to be confirmed.

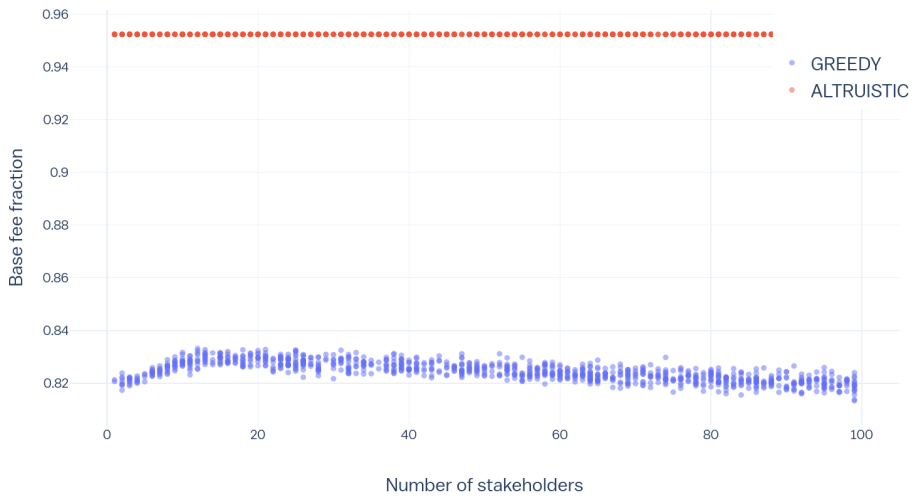


Figure 4.4: Base fee ratio for standard strategies

We see that Altruistic maintains an extremely high base fee ratio close to 1, since in this strategy proposers will not wait for higher tips before confirming transactions. Greedy still maintains a high ratio of around 0.82-0.84. Furthermore, this ratio is very collusion-insensitive — it does not substantially change no matter how many stakeholders participate. Thus, even if all participants follow a short-sighted greedy algorithm for picking transactions, users can still use a simple bidding strategy of, say, bidding 30% more than the base fee.

Figure 4.5 shows the fee multiplier at the end of each simulation. We see that with both strategies, the fee multiplier ends up within a tight range — most likely, the revenue-maximizing point. This is despite the fact that HillClimb is not very accurate at finding the revenue-maximizing point, especially when many stakeholders exist, due to its randomized nature.

Greediness does not pay off We now show that in a one-on-one contest, Greedy receives similar profits to Altruistic. We run a longer, 10000-block simulation, except this time we have one stakeholder following Altruistic and one stakeholder following Greedy.

We visualize a typical run of the simulation in Figure 4.6, which plots the revenue (averaged over 100 blocks) of both stakeholders, as well as the fee multiplier. We see that Greedy does not, in fact, gain more revenue than Altruistic. Intuitively, this is because when a Greedy proposer refuses to accept a transaction, and its tip is therefore forced upwards, the increased tip actually goes to the

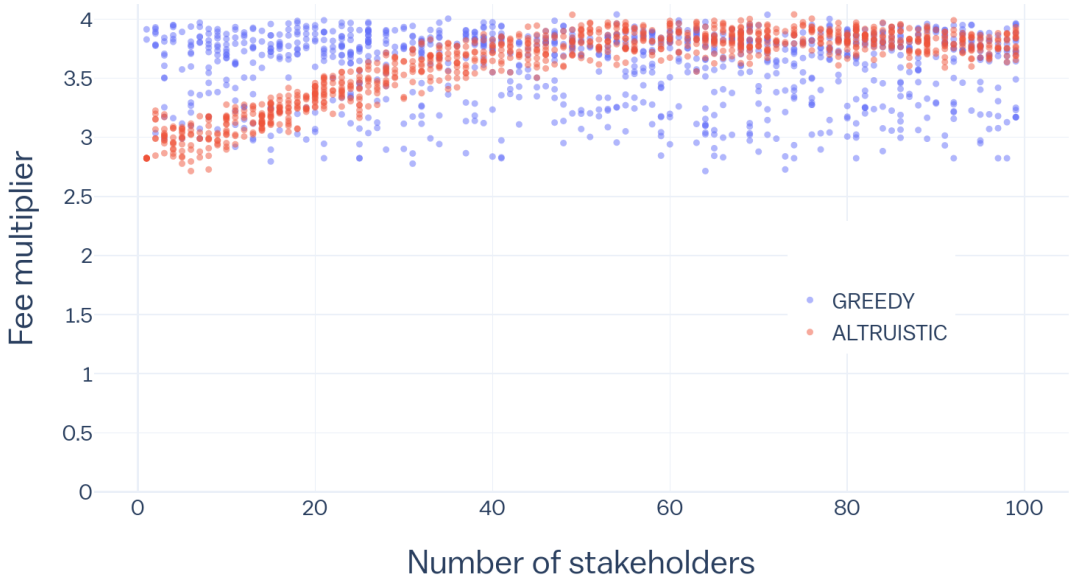


Figure 4.5: Fee multiplier for standard strategies

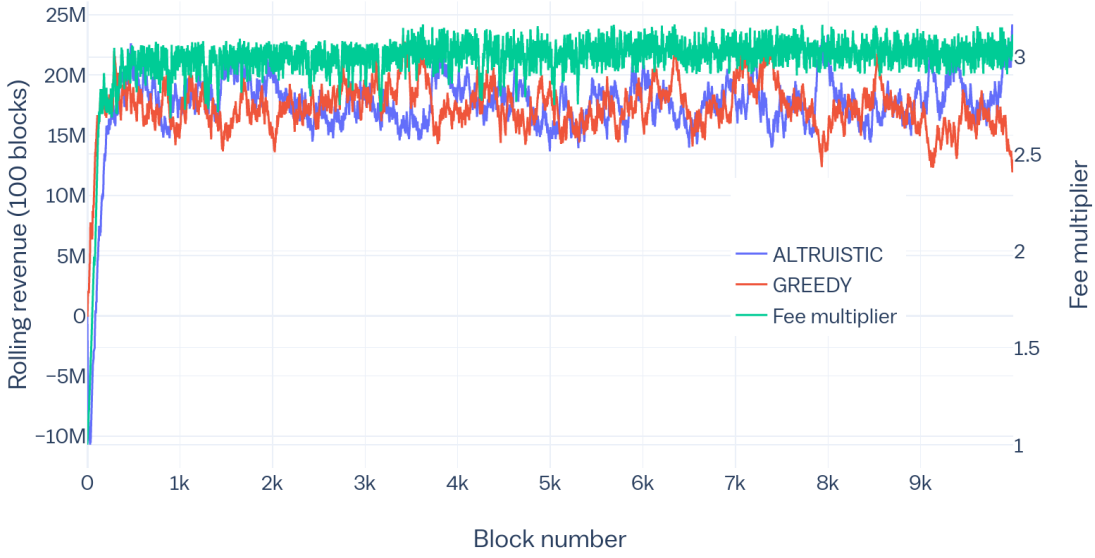


Figure 4.6: Revenue for Greedy vs Altruistic

next proposer, who may be following an Altruistic strategy. Thus, Greedy's efforts to increase tips end up being "altruistic".

4.5.3 Pathological strategies

We now look at the pathological strategies Lazy and Monopolist, showing that they always lose against standard strategies.

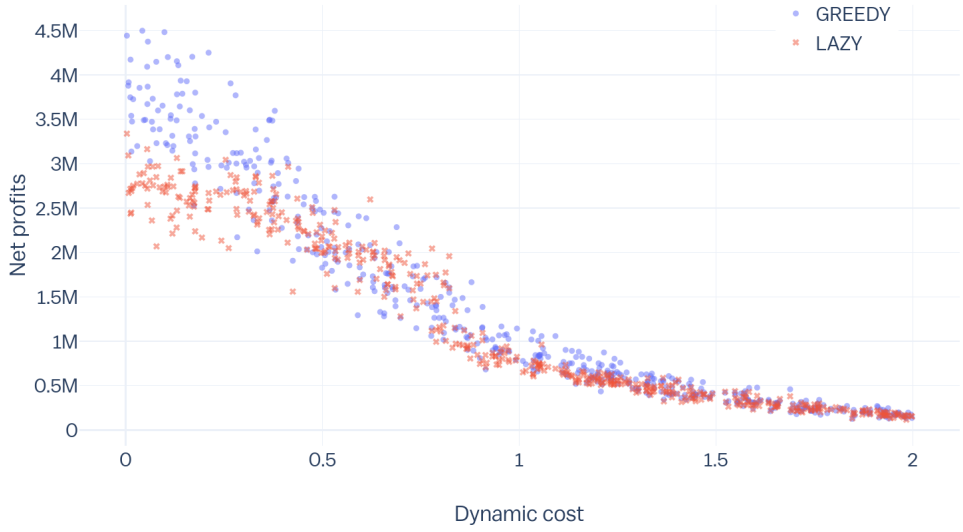


Figure 4.7: Lazy vs Greedy

Laziness does not pay off We first investigate the case where Lazy faces off head-to-head against Greedy. We run a 1000-block simulation, varying S.dyncost between 0 to 2 to represent differing opportunity costs for including a transaction. The results are shown in Figure 4.7 We see that at low costs, Greedy is much more profitable — the profit gained by normally accepting transactions more than offsets their costs. Even at higher costs, Greedy is still marginally better, since Lazy rejects even transactions that would have been profitable. Thus, a rational stakeholder would not choose Lazy as a strategy.

Monopolists fail We now investigate Monopolist, a strategy that picks all transactions, hoping to deny others revenue. We run a simulation analogous to our previous simulation, and plot the results in Section 4.8. We see that Monopolist is a strategy that's just as bad. At lower costs, it is

basically equivalent to Greedy, since most transactions pay more fees than their cost. At higher costs, all Monopolist accomplishes is to waste money confirming transactions that pay low fees. Furthermore, comparing with Fig 4.7, we see that Monopolist does not accomplish the goal of reducing Greedy's revenue at any cost.

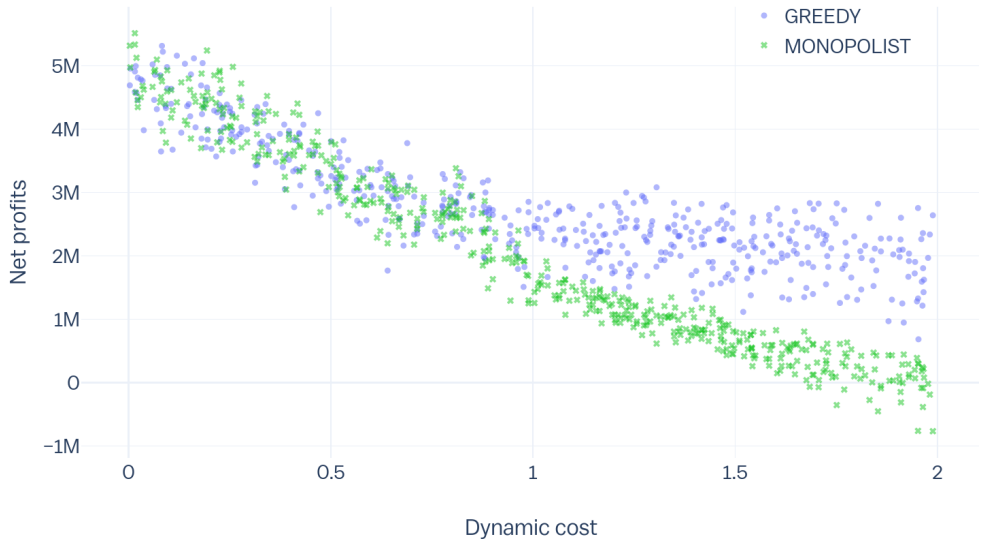


Figure 4.8: Monopolist vs Greedy

4.5.4 Failure of alternative fee models

Finally, we show that Synkletos's "cartelizing" feature of collectively setting a base fee that is then split is crucial to stability, despite it appearing to be unfair and monopolistic. To do so, we compare our model of Synkletos to an alternative model where instead of accumulating in the pool, fees are burned. This results in a system very similar to Ethereum's EIP-1559 fee economy [But+14b].

We repeatedly run a 1000-block simulation with different numbers of Greedy stakeholders, comparing the outcome of EIP-1559's fee economy with that of Synkletos. Results are plotted in Figure 4.9. We see that unlike Synkletos, which produces a stable fee multiplier regardless of the number of stakeholders, EIP-1559 suffers from two instabilities. First, at very low numbers of stakeholders (i.e. high levels of collusion), it is no longer in anyone's interest to correctly vote for the fee multiplier. Instead, everyone benefits from a low fee mulitplier that causes most fees to turn into tips. Secondly, at very high numbers HillClimb no longer converges, since the lack of a fee pool

means revenue for stakeholders is extremely noisy. Most importantly, we see that Synkletos does not charge significantly more fees than EIP-1559.

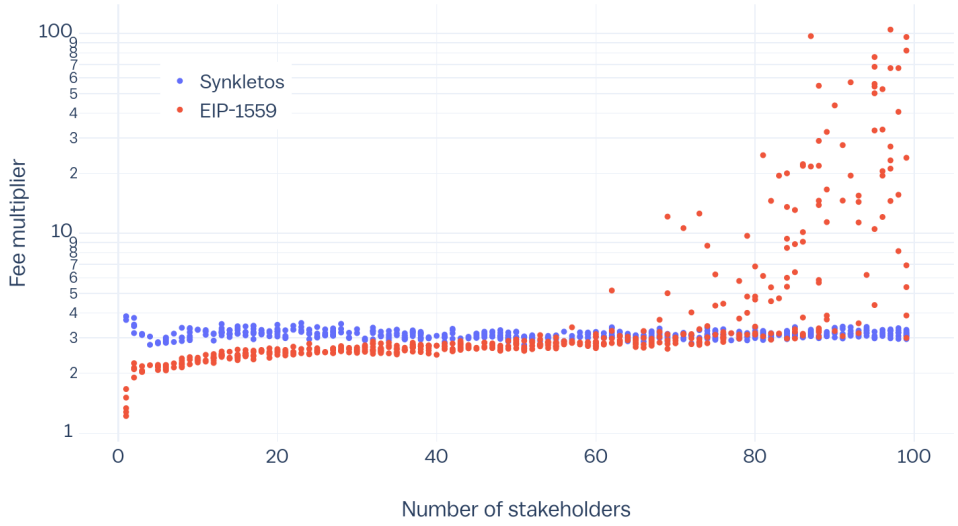


Figure 4.9: EIP-1995 vs Synkletos

4.6 Conclusion

In this chapter we described Synkletos, the cryptoeconomic system behind consensus in Themelio. We first argued that instead of aiming for some sort of ideal outcome and trying to design a cryptoeconomic mechanism with just the right incentives to produce that outcome, we should aim for "despot simulation" — having the blockchain function as if controlled by a rational, profit-seeking centralized entity. This allows us to design a mechanism that, unlike existing blockchain consensus mechanisms, is extremely resistant to collusion and provides for a much more user-friendly fee market.

We then validate our results by a stochastic agent-based simulation, showing that Synkletos reliably simulates a despot at varying levels of collusion, while resisting adversarial strategies. We also see that non-despot-simulating strategies like EIP-1559 do not produce stable behavior and can fail catastrophically when stakeholders collude.

Chapter 5

Melmint: low-volatility decentralized cryptocurrency

5.1 Introduction

All decentralized blockchains, starting from Bitcoin [Nak08], are intimately tied to on-chain cryptocurrencies. These cryptocurrencies, in addition to serving as protocol-internal units of value for incentive design, also provide a trustless and irreversible payment medium to end users. Many merchants and payment processors accept cryptocurrencies for this reason, and cryptocurrency trading has become a significant financial market with a combined market capitalization of over 100 billion US$ [Coi].

Despite their rising popularity, however, blockchain cryptocurrencies do not actually see a lot of usage as *money* — an asset that's simultaneously a store of value, unit of account, and medium of exchange [Jev85]. Cryptocurrency payment processors (such as BitPay [Bit19a]) typically convert payments immediately into fiat currencies, few people store personal savings in cryptocurrency, and prices are generally not quoted in cryptocurrency terms. At best, cryptocurrency is used as a "hot-potato" payment intermediary; at worst, it is used entirely as a speculative asset sitting on exchanges.

This state of affairs is unacceptable, especially for Themelio. Without a usable monetary unit, not only will it be difficult to build robust applications on top of Themelio, mechanism design becomes impossible if nothing on-chain is reliably valuable. Current cryptocurrencies lack "moneyness" mostly due to their *extremely volatile value*. Cryptocurrency exchange rates can fluctuate as much as 15% in a single day [Coi], greatly increasing the risk of long-term holding and hindering usage as

money. Volatility is in turn caused by entirely demand-agnostic currency issuance — for example, Ethereum simply mines 2 ETH per block [Woo14] — which causes the fluctuating demand of cryptocurrencies to directly translate into large changes in price.

Much of the existing work on solving this problem focuses on *stablecoins*, or cryptocurrencies that are pegged to an external value-stable asset, generally a fiat currency like the US dollar. Stablecoin schemes include centralized currencies like Tether [Tet] and TrueUSD [Tru] that act as fiat-denominated IOUs against a trusted bank as well as semi-decentralized systems such as MakerDAO [Tea17] which attempt to hold a peg through algorithmic monetary policy involving complex on-chain financial assets.

A problem common to all stablecoins targeting an exchange rate to an asset external to the blockchain, though, is that there is no way of measuring this value on the blockchain without abandoning endogenous trust. All stablecoins, even "decentralized" ones like MakerDAO, rely on trusted *price oracles*. In addition, coins tied to external assets are inherently vulnerable to shocks in the value of that external asset — USD-pegged stablecoins, for instance, can only be as stable as the US dollar.

In this part, we present Melmint, Themelio's cryptocurrency issuance scheme. Melmint is the first attempt to create an *endogenous* value-stable cryptocurrency issuance scheme we are aware of in the literature. Melmint creates a synthetic asset called the *mel*, pegged to a hypothetical unit of account known as the DOSC, indexed to the value of one *d*ay *o*f *s*equential *c*omputation on an up-to-date processor. DOSCs have remained surprisingly stable in value even though processor speeds have increased several orders of magnitude over the past few decades, making it one of the only scarce assets we know that are both value-stable and measurable without external oracles.

Melmint is composed of two components. First we utilize a mechanism known as Elasticoin [DB19], where we use non-interactive proofs of sequential work [CP18] to trustlessly create an asset — the *erg* — a defined sum of which has a *minting cost* equal to 1 day of sequential computation. Then, we design a monetary control loop that interacts both with the erg and with Themelio's proof-of-stake token, the *sym*, to mint a value-stable *mel*, by pegging 1 mel to 1 DOSC worth of sym. Melmint allows Themelio's base currency to maintain a long-term stable value without any trusted issuers or data feeds, solving a major open problem in cryptocurrency design.

5.2 Background and motivation

In this section, we take a look at the background of the cryptocurrency volatility problem. We first examine why cryptocurrency prices are so volatile and some less obvious problems this volatility causes, and then we take a look at existing work that attempts to stabilize cryptocurrency values.

Finally, we argue that existing approaches are all inadequate and that a new mechanism is badly needed.

5.2.1 The problem of volatility

Ever since their inception, cryptocurrencies have been exceptionally volatile. In fact, they're probably some of the most volatile non-derivative financial assets in existence — Bitcoin on average fluctuates by more than 3% every day [Bit19b], orders of magnitude higher than fiat currencies, even though it has by far the most market liquidity of any cryptocurrency.

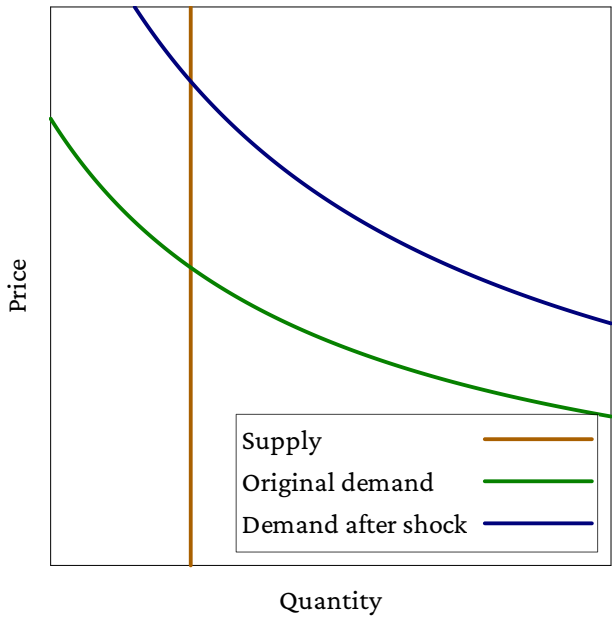


Figure 5.1: Inelastic cryptocurrency supply

Such extreme volatility is due to a combination of volatile demand and *perfectly inelastic supply*. Cryptocurrencies generally are issued on fixed schedules that ignore market conditions, leading to a situation illustrated in Figure 5.1: changes in demand cause sudden and large changes in price. However, volatility causes two serious problems:

- *The cryptocurrency ceases to be useful as money*. Since cryptocurrency units no longer have a stable purchasing power, they cannot fulfill the duties of a currency well. It becomes impractical to do business, store wealth, etc. denominated in a cryptocurrency. High exchange-rate risk also greatly increases transaction costs of cryptocurrency-denominated decentralized

finance (DeFi) applications. This is a well-acknowledged problem [But14; Iwa+14; Sam19] with volatile cryptocurrencies.

- *Cryptoeconomic mechanisms are destabilized*. More insidiously, with a volatile on-chain unit of value, it is much harder to design safe cryptoeconomic systems. For example, when analyzing Bitcoin's security it is common to assume that a rational self-interested entity wants to maximize profits denominated in bitcoins [ES14; SSZ16]. Yet with almost every action inscrutably influencing the price of a bitcoin one way or another, it is very hard to be sure of any cryptoeconomic proofs. It also makes it hard to design cryptoeconomic rewards with defined sizes, an approach sometimes considered neglected [But18a] in current designs.

Thus, volatility is neither a transient issue due to volatile cryptocurrency adoption nor a small inconvenience that can easily be abstracted away. We believe that eliminating the extreme volatility of cryptocurrencies is crucial to their long-term success.

5.2.2 Externally-pegged stablecoins

Existing approaches to creating a stable cryptocurrency generally focus on pegging it to a real-world asset, generally the US dollar. In fact, "stablecoins" are generally defined simply as on-chain assets pegged to real-world ones. Let us examine some existing approaches to creating such pegged stablecoins, and why they will not work for Themelio.

Centralized currency boards The most straightforward family of stablecoins uses a trusted bank that promises to exchange each unit of on-chain cryptocurrency for a fixed amount of the off-chain asset to which the peg is maintained. Stablecoins in this category include Tether [Tet], TrueUSD [Tru], and many others. This arrangement is known as a *currency board* system, and it is used by many robustly pegged fiat currencies [Wal89]: the Hong Kong dollar, for example, is essentially an IOU issued by the Hong Kong Monetary Authority for 0.128 US dollars.

Currency board stablecoins have an advantage in that as long as the bank is trustworthy, no economic shock of any size can disturb the peg. Even if all users suddenly dump the pegged coin, the bank always has enough assets to sell to maintain the peg. Unfortunately, such stablecoins suffer from the obvious flaw of *counterparty risk* — if the institution providing the backing is untrustworthy, everything collapses. This is not a far-fetched possibility: unsound fiat currency boards such as that of the Argentinian peso [HM02] have undergone total collapses, and the risk of unsound backing [GS18] is a significant factor hindering the adoption of Tether.

"Decentralized" stablecoins Many other stablecoins projects exist that eliminate counterparty risk altogether by eschewing a trusted bank. They instead use some form of *on-chain algorithmic monetary policy*: a control loop typically implemented in a smart contract autonomously adjusts the money supply to target an exchange rate. No trust is required in any centralized issuer to achieve a stable peg. The exact mechanism used varies wildly from system to system; the most popular such stablecoin, MakerDAO [Tea17], uses a sophisticated mechanism centered around maintaining reserves of significantly more than $1 worth of ETH for every $1 coin issued so that the peg can be maintained even when drops in the value of ETH wipe out a large percentage of the value of the reserves.

There are two significant problems with all non-currency-board stablecoin proposals though. First, *issuing an asset A pegged to asset B without holding asset B* is profoundly difficult, yet such algorithmic stablecoins must be able to peg a currency to, say, US dollars without the ability to hold any dollars. The challenge is comparable to that of a central bank attempting to peg a currency to a foreign currency without any foreign exchange reserves, or a commercial bank investing depositors' dollars entirely in assets like commodities and foreign-currency bonds whose values are decoupled from that of the dollar. A way of doing either task safely would prove very profitable in the existing financial world; the fact that nobody engages in such business is perhaps evidence that it is in some way uninsurably risky. In practice, we have indeed seen spectacular collapses in algorithmic stablecoins with unsound collateral design, such as TerraUSD. [Bri+23]

More importantly, even stablecoins without a central issuer require trusted *oracles* to feed in the current price of the stablecoin, which is crucial to driving the algorithmic monetary policy. Although mitigations such as using the median of multiple oracles do exist, measuring facts external to the cryptoeconomic mechanism — the "oracle problem" — is one of the major fundamental issues with smart contracts in general [Zhe+18]. Attempts at making decentralized oracles, such as SchellingCoin, typically fall to clever game-theoretical attacks that collapse their security entirely.

Thus, both currency board and "decentralized" stablecoins fundamentally still rely on centralized trust. We conclude pegging to blockchain-external currencies is unlikely to be a viable path to a low-volatility cryptocurrency with endogenous trust.

5.3 Tracking the DOSC with Elasticoin

5.3.1 Why Elasticoin?

There is a somewhat surprising lack of detailed proposals for endogenously stabilizing a cryptocurrency's value in the literature. For many years, the only things we had were vague suggestions of

econometrically measuring on-chain activity [Iwa+14; Wik] to calculate a desired money supply. The most concrete proposal was probably a blog post by Vitalik Buterin [But14] that attempted to construct a complex model to deduce the dollar-denominated price of Bitcoin from blockchain-endogenous metrics. Even there, he admits that the model can probably be gamed.

We developed Elasticoin [DB19] as the first detailed proposal for trustlessly reducing cryptocurrency volatility. Its core concept is to *fix the cost of minting a coin* to that of a certain quantity of "wasted" sequential computation time on the fastest processor available. Such a "proof of wasted time" can be trustlessly validated by combining non-interactive proofs of sequential work [CP18] with a continually updated on-chain speed record.

Figure 5.2 illustrates the supply and demand curves for newly-issued Elasticoin. Supply is very elastic, since whenever demand pushes the price of a coin above the cost of creating it, anybody can mint coins and take a risk-free profit. This one-sided arbitrage effectively establishes a limit to the price for the issued cryptocurrency.

Elasticoin reduces volatility in two ways. The most obvious one is that broadly stagnant or growing demand will result in a stable price very close to the cost of minting. Less obviously but far more importantly, Elasticoin *cuts off speculative demand*.

Cryptocurrency demand has been analyzed as broadly consisting of two parts: transactional demand CD_T from people seeking to use the currency to buy goods or hold as a short-term store of value, added to speculative demand CD_S based on rational expectations of higher values in the future. Anecdotally, demand for most cryptocurrencies is dominated by CD_S, but with Elasticoin, $CD_S \approx 0$ in a steady-state economy because there is no expectation of higher future values at all.

Thus, Elasticoin both flattens the supply curve and dampens movements in the demand curve, achieving low volatility in normal conditions without stablecoin-like oracles or financial instruments. This is insufficient to create a truly stable cryptocurrency that has rock-solid value even in abnormal economic environments, but Elasticoin forms a crucial component of Melmint's design.

5.3.2 An overview of Elasticoin

We go on a high-level tour of Elasticoin. This is essentially an abridged version of our previous work. [DB19]

Non-interactive proofs of sequential work Elasticoin relies on *non-interactive proofs of sequential work* (NiPoSW), a primitive we built by applying the Fiat-Shamir heuristic to Cohen and Pietrzak's "Simple Proofs of Sequential Work" (SPoSW) [CP18].

SPoSW's construction is based upon building a hash DAG, but the specifics are not really important for the purposes of understanding NiPoSW. In SPoSW, a prover P interactively proves to a verifier V that sequential work proportional to $N = 2^n$ was done. SPoSW consists of the following steps, executed using cryptographic functions defined in [CP18] we call PoSW, open, and verify:

- *Statement*: In the first step of SPoSW, $\mathcal{V}$ samples a random binary string χ, known as the "statement", sending it to $\mathcal{P}$.
- *Compute*: After Statement, $\mathcal{P}$ computes (by doing N sequential operations) a proof $(\phi, \phi_{\mathcal{P}}) := \mathsf{PoSW}(\chi, N)$. ϕ is sent to $\mathcal{V}$, but $\phi_{\mathcal{P}}$ is kept locally at $\mathcal{P}$
- *Challenge*: Then, $\mathcal{V}$ challenges $\mathcal{P}$ with a random challenge $\gamma = (\gamma_1, \ldots, \gamma_t)$.
- *Open*: $\mathcal{P}$ responds to this challenge with $\tau := \mathsf{open}(\chi, N, \phi_{\mathcal{P}}, \gamma)$ and sends it to $\mathcal{V}$.
- *Verify*: $\mathcal{V}$ computes and outputs $\mathsf{verify}(\chi, N, \phi, \gamma, \tau) \in \{\mathrm{accept}, \mathrm{reject}\}$

We use the well-known Fiat-Shamir heuristic [FS86] to transform SPoSW to its non-interactive version, NiPoSW, which exposes only two steps:

- *Solve*: $\mathcal{P}$ computes $\phi = \mathsf{Solve}(\chi, n)$, where χ is a random "statement" that *cannot be influenced by* $\mathcal{P}$, n is a difficulty parameter indicating that $N = 2^n$ must be done, and ϕ is the resulting proof of sequential work. $\mathcal{V}$ then broadcasts ϕ.
- *Verify*: Any $\mathcal{V}$ can then run $\mathsf{Verify}(\phi, \chi, n) \in \{\mathrm{accept}, \mathrm{reject}\}$ to check the validity of the proof.

More details can be found in our previous work [DB19] on Elasticoin.

Minting algorithm Now that we have a primitive for publicly proving completion of sequential work, we can now construct Elasticoin's minting algorithm. Unlike most cryptocurrency issuance protocols, Elasticoin minting is a two-step process, where minters first register puzzles on the blockchain to later solve with a separate transaction.

To register a puzzle, a minter broadcasts a specially formatted transaction containing $\mathcal{A}$, the identity of the minter, generally a public key or other form of cryptocurrency "address". In Themelio's case $\mathcal{A}$ is a hash of a Melodeon contract. Once the transaction is confirmed on the blockchain, the minter notes t, the block height at which the transaction is embedded, and calculates

$$\chi_A = \mathcal{A}||R_t$$

where R_t is a public value unpredictable until the creation of block t. In Themelio, that's the block header hash.

The minter then begins work on a puzzle with seed $\chi_\mathcal{A}$ and a difficulty parameter n of her own choice. After solving the puzzle, she broadcasts another special transaction containing the solution to the puzzle,

$$\phi = \mathsf{Solve}(\chi_\mathcal{A}, n)$$

which entitles her to claim a reward $r(n)$.

Through this process, the minter publicly proves that after the puzzle is registered and before it is solved, she finished $O(2^n)$ sequential operations. She cannot cheat by precomputing a solution, as $\chi_\mathcal{A}$ cannot be calculated in advance.[1]

We now look at how rewards are determined; Elasticoin's reward function is what powers the "magic" that drives its minting cost to track sequential computation time. We keep track of a variable $v_{\max}$, representing "the speed of the fastest minter ever observed on the blockchain, measured in operations per day". Every time a solution transaction is posted to the blockchain, at block height t_{solve}, solving a puzzle posted at height t_{register} with difficulty parameter n (showing proof of doing $O(2^n)$ work), we update $v_{\max}$:

$$v_{\max} \leftarrow \max(v_{\max}, v)$$

where

$$v = \frac{C \cdot 2^n}{t_{\text{solve}} - t_{\text{register}}}$$

where C is the number of blocks produced in a day — for Themelio, $C = 2880$.

Rewards are then calculated as

$$r(n) = \frac{2^n}{v_{\max}} \cdot \frac{v}{v_{\max}}$$

which is intuitively the total number of "fastest processor days" 2^n hashes is equivalent to, multiplied by a factor that linearly penalizes minters that are slower than the fastest processor.

[1] This is the case unless a quorum of stakeholders colludes in consensus and precomputes a long list of block hashes in advance. But such a quorum would be able to do much more damage to Melmint, for example by delaying transactions or reducing the block interval so that minter speeds seem slower than reality, or simply by executing a number of attacks on the consensus itself.

Incentives and security Given that $v_{\max}$ is a reasonable estimate of the amount of sequential work the fastest processor generally available can do, it is obvious that for the fastest processor, minting one full coin requires a whole day of sequential work, achieving our goal. Why, however, will $v_{\max}$ be a good estimate? And will processors slower than the very fastest get reasonable rewards?

Importantly, $v_{\max}$ never decreases; we assume that technology never goes backwards and recent processors are at least as fast as old ones. Therefore, we can dismiss attacks attempting to reduce $v_{\max}$. On the other hand, faking a high value of $v_{\max}$ is impossible without actual advances in computational speed, as puzzles cannot be precomputed. Finally, attempting to prevent $v_{\max}$ from changing is also practically impossible. Even if almost all the minters form a cartel and solve puzzles purposefully slowly, one honest minter is enough to keep $v_{\max}$ correct.

Processors slower than $v_{\max}$ take longer to solve puzzles of the same difficulty, yet are doubly penalized by the $v/v_{\max}$ term. This intentionally disincentivizes minting using suboptimal hardware, leading to emergent standardization on the most efficient way to mint at any given time, reducing market uncertainty about the cost of minting and thus the price. Furthermore, the term all but eliminates the possibility of using slow computation with volatile but usually cheap prices, such as spare computational cycles or botnets, further reducing volatility.

5.3.3 Supply elasticity is not enough

The major problem with Elasticoin is that even though supply elastically expands when demand for currency is high, when demand is low supply cannot contract. This is illustrated by the "knee" in the supply curve in Figure 5.2. In fact, starting from a steady state where quantity supplied matches quantity demanded and the price is close to the ceiling, any drop in demand will cause a commensurate drop in price. Furthermore, if economic shocks cause demand to suddenly decrease, the price may become so far away from the minting cost that even the CD_S-damping effects of Elasticoin become irrelevant. Elasticoin's volatility would simply degenerate to be similar to that of a traditional cryptocurrency.

Elasticoin does make an important contribution in creating a *one-way* "peg" between a cryptoasset and a trustlessly measurable value unit, but it is clear that a different approach is needed to truly achieve our goal of a trustless stable cryptocurrency. Specifically, not only do we need high supply elasticity for newly minted coins, but also a way of reducing the number of coins in circulation when there's no demand for new coins.

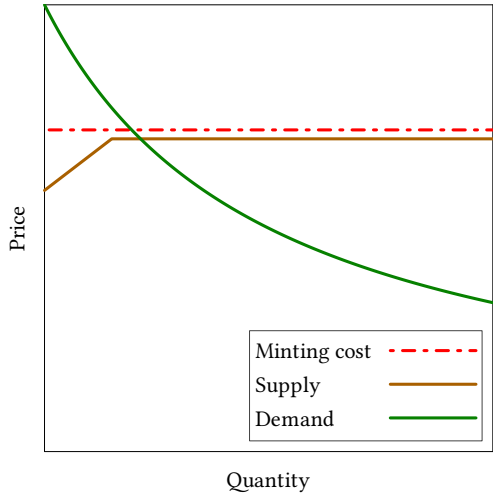


Figure 5.2: Supply and demand of an Elasticoin-issued currency

5.4 Melmint's core mechanism

We will now discuss the design of Melmint, our solution to the trustless stable cryptocurrency problem. We describe an instance of Elasticoin we use for establishing a "temporary" asset, the *erg*, tracking the DOSC (day of sequential computation). Then, we detail the central control loop of Melmint, based on an automated auction of syms (proof-of-stake tokens) for DOSCs and a two-way peg between 1 mel (stablecoin) and 1 DOSC worth of syms. Figure 5.3 gives an overview of Melmint's design.

5.4.1 Context: the Themelio blockchain

As its name suggests, we created Melmint as Themelio's mel-minting procedure. It is, however, portable to many other blockchains, as we will discuss in Section 5.4.4.

5.4.2 Establishing a trustless value unit

As a building block for Melmint, we introduce a new built-in cryptocurrency, the "erg". Ergs are created using the Elasticoin algorithm, but instead of having a fixed minting cost, we create k ergs per 24 hours of sequential computation (a DOSC), where k is an exponentially increasing constant that

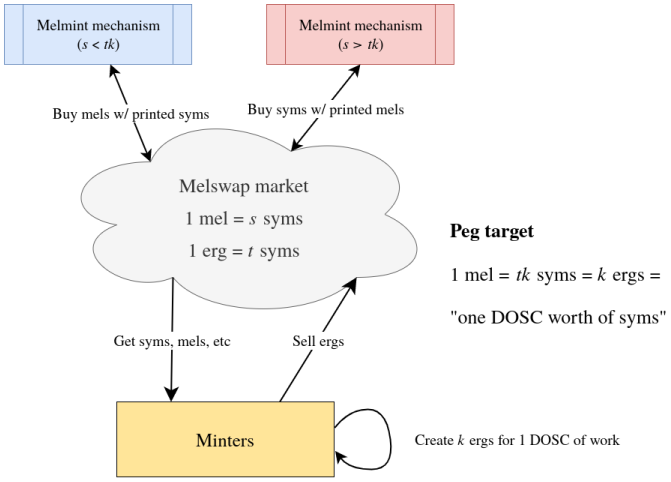


Figure 5.3: A simplified overview of Melmint

starts at 1 at the first block, but increases by 0.00005% every block. [2] The purpose of an exponentially increasing k is so that *the supply of ergs is dominated by freshly minted ergs* — ensuring that the *market value* of k ergs tracks the 1 DOSC *cost of creating* those ergs.[3] If k were instead constant, a glut of past minted ergs would decouple their market value from their creation cost, leading to the same problem mentioned in Section 5.3.3.

Thus, the erg is a "perishable" asset utterly useless as money, but in exchange, the current value of k ergs cannot deviate far from the cost of a "day of sequential computation" of the most efficient minter. This is an important property that we will exploit in designing Melmint's core mechanism.

5.4.3 Melmint's core mechanism

The main objective of Melmint's core mechanism is to hold the price of 1 mel around 1 DOSC. This is accomplished by a feedback loop that interacts with *Melswap*, a Uniswap-like constant-product decentralized exchange built into the blockchain protocol that supports all Themelio-based tokens.

[2] This is an arbitrary "magic number". There is no significance behind the specific value, except that it needs to be big enough to be much greater than any realistic real interest rate (and thus destroy any "hoarding" value), but small enough to not cause integer overflow in the lifetime of the blockchain.

[3] Or more precisely, the *market clearing price of 1 day of sequential computation*. Actual costs for computation are likely to be very producer-dependent — e.g. there's likely a small amount of idle computation that is "free" — but an open market should settle around a unified, nonzero, sustainable cost, since sequential computation is a scarce resource.

Erg-sym auction We continually auction newly created syms for ergs to establish the exchange rate between ergs and syms. This is done by simply directing half of Sym inflation into the Melswap sym/erg market, subsidizing its liquidity and incentivizing people to create ergs to exchange for freshly-inflating syms.

This gets us the erg/sym exchange rate: 1 ergs to t syms. Note that because k ergs must be produced by a day's worth of computation, this essentially tracks the exchange rate between 1 sym and the market-clearing price of 1 day of computation — 1 DOSC. [4]

Stabilizing mel value With the erg/sym exchange rate t determined, we can now stabilize the value of the mel. This is done by intervening in the mel/sym market. We want to guarantee that anyone can exchange 1 DOSC — k ergs — worth of syms to obtain 1 mel, or destroy 1 mel to obtain 1 DOSC worth of new syms. This is effective because syms have an independent source of economic value: they can be freely staked to capture fees from on-chain transaction activity. [5]

More specifically, given a mel/sym exchange rate of 1 mel to s syms, we want to target an exchange rate of $s = tk$, where k is the erg-DOSC conversion factor. This is done through a procedure that either prints mels to buy syms, or vice versa, at every block height, with two cases:

- $s < tk$: In this case, Melmint continually prints syms out of thin air, using them to buy mels, which are subsequently destroyed. This artificially increases the demand for mel, increasing its purchasing power until $s = tk$
- $s > tk$: In this case, Melmint instead prints mels, using them to buy syms that are then destroyed. This increases the supply of mels, decreasing its price until $s = tk$.

In addition, the maximum amount of syms or mels printed each block is *proportional to a lower bound on total market liquidity* of the sym/mel pair, measured through Melswap, an on-chain decentralized exchange.[6] This ensures that Melmint will neither overreact to an illiquid mel/sym market nor be too weak to control a highly liquid mel/sym market.

[4] Unsustainably "free" computation, such as idle computation, will quickly be consumed by minting activity and become scarce and expensive, so they are unlikely to cause large deviations in the price feed.

[5] In this way, unstaked syms have value backed by their potential to be staked. Given the relative ease syms can be staked and unstaked, the market value of unstaked sym should closely track its income potential if staked.

[6] This is just a lower bound because we do not include off-chain liquidity, but that is sufficient to ensure stability.

Discussion Taken as a whole, Melmint effectively pegs each mel to 1 DOSC worth of syms, stabilizing the price of 1 mel to around the cost of wasting one day of sequential computation to create 1 DOSC. This is because arbitrage opportunities exist that push mel prices towards 1 DOSC no matter whether mels are too expensive or too cheap.

During periods of increasing demand, the price of a mel is above tk syms, at which point Melmint will begin selling mels and buying syms on Melswap. This increases the supply of mels until the price decreases, establishing an Elasticoin-like hard ceiling on the mel price.

When demand decreases, mels may depreciate until they are no longer worth tk syms. At this point, Melmint instead contracts the supply of mels by buying them up with freshly printed syms. This process will be repeated until enough mels are destroyed that the price increases back towards 1 DOSC.

One important observation is that through its main mechanism, *Melmint backs the value of a mel by expropriating sym holders*. When new syms are created in exchange for destroying mels, this directly dilutes the value of one sym. In the long run this is hopefully balanced by the opposite process destroying syms and raising their value, but in any case this means that people holding syms contribute reserve capital to back the mel, and the total market capitalization of the sym is a good estimate of the "implicit reserves" that Melmint has to defend the peg. Fortunately, we will show in Section 5.5.1 that these reserves are almost certainly many times the amount of circulating mels, ensuring the stability of the mel-DOSC peg.

5.4.4 Porting to other blockchains

Given the almost entirely blockchain-agnostic description of Melmint above, it is straightforward to implement Melmint on smart-contract blockchains such as Ethereum and EOS. None of the core functionality of Melmint uses any blockchain-specific "black magic".

The main subtlety is the definition of a sym: syms are intimately tied to consensus participation and stakeholder rewards in Themelio, while a non-Themelio deployment of Melmint is clearly unable to issue any cryptocurrency with such powers. A "useless" sym with no inherent value will not work, as such a token would not provide nearly enough implicit reserves.

The most important property of the sym in Themelio is that its value is largely based on revenues from transaction fees, and thus is proportional to the total economic value transacted within the mel-using ecosystem as a whole. As we will see in Section 5.5.1, this *fee-based sym valuation* is crucial to Melmint's robust stability.

Fortunately, replicating this on other blockchains is fairly easy: one can simply have sym-holders split a small percentage fee on each mel-denominated transaction to simulate Themelio's transaction

fees. This will then make the total market capitalization of syms proportional to mel-denominated economic activity.

5.5 Evaluation

In this section, we first analyze the stability of the system using both qualitative arguments and quantitative data from real-life financial markets. We then examine the resistance of the system against attack and the cryptoeconomic incentives involved. Finally, we compare Melmint to the existing literature.

5.5.1 Stability of implicit reserves

We start with a rough but conservative heuristic analysis of Melmint's stability. As we have previously mentioned, the market cap of syms acts as an implicit reserve that is drained when syms are inflated to buy and destroy mels. Thus, the ratio of the total value of all circulating syms to that of all circulating mels — the *implicit reserve ratio* — must be above 1 to guarantee stability.

Let us estimate what this ratio would be in a realistic blockchain economy. Syms derive their value by "taxing" mel transaction activity through fees, block rewards, etc. This process generally extracts some small fraction of the total economic value transacted in mels. In Bitcoin, the proportion of the total transaction volume captured as miner revenue is around 2-10%, a number curiously similar to the percentage of GDP raised by a wide range of premodern taxes on vital commodities, such as salt taxes in imperial China [Feu84]. As a safe estimate, let us assume that the revenue r captured by sym-holders is 2% of mel-denominated economic activity Y: $r = 0.02Y$.

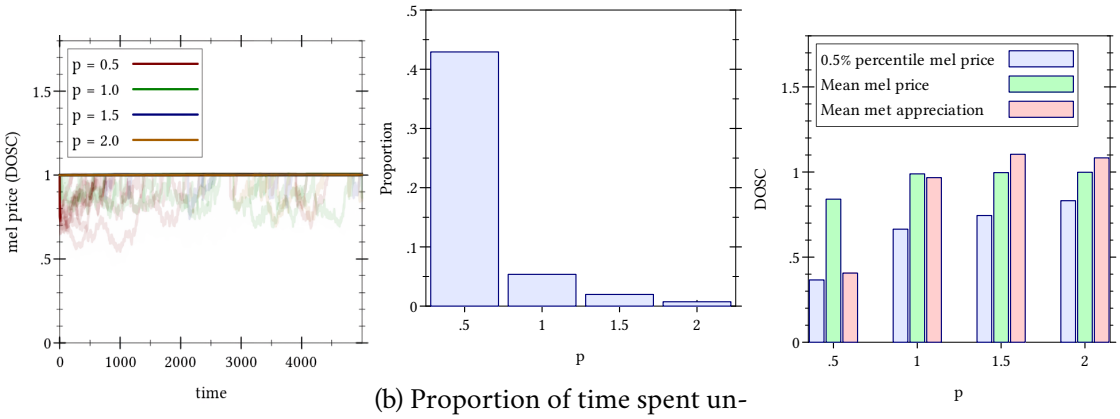
We can now estimate the market capitalization of syms through a discounted cash flow model: given a discount rate of d, the total value of all syms Θ would be:

$$\Theta = \sum_{i=0}^{\infty} r(1-d)^i = \frac{0.02Y}{d}$$

Assuming a typical discount rate of $d = 0.03$, this gives $\Theta = 0.67Y$. This is then also the upper limit of the total value of syms we can safely issue. But examining existing economies, we see that $0.67Y$ is generally well above the amount of currency in circulation. In Table 5.1, we list the ratio of money supply to annual economic activity for the US economy and for Bitcoin. For the US, we use the M1/GDP ratio, while for Bitcoin we divide the total number of bitcoins by 360 times the

Economy	5% percentile	Median	95% percentile
USA (since 1960)	0.105	0.149	0.245
Bitcoin (since 2014)	0.107	0.228	0.495

Table 5.1: Ratio of money supply to yearly economic activity



(a) 10 runs of the simulation

(b) Proportion of time spent unpegged

(c) General stability statistics

Figure 5.4: Market simulation results

daily on-chain transaction volume. We see that in both cases the ratio is well below our conservative estimate of 0.67.

Thus, we expect that Melmint's implicit reserves are probably sufficient even to withstand large "runs" on the mel.

5.5.2 Stability of DOSC value

We now move on to an analysis of the value of a DOSC. The key conjecture behind Melmint, as well as its predecessor Elasticoin, is the value-stability of a *DOSC* (day of sequential computation). The DOSC is informally defined as the cost of 24 hours of sequential computation done on an *up-to-date* processor. Anecdotally, this seems likely to be true — single cores rented out at cloud computing providers seem to have cost 5-10 USD a month for at least a decade.

Unfortunately, a rigorous analysis is quite difficult. This is because "cost of running for a day" is not a metric people typically gather statistics on. Instead, "cost of computing X cycles" (dollars per million floating-point operations, etc.) is usually what people care about. Thus, as far as we know,

Figure 5.5: DOSC speed over time

we cannot directly read the value of a DOSC off of any kind of aggregate data source. Instead, we must indirectly calculate it based on performance and price data of historical CPUs.

There are two separate questions to answer in order to determine the historical value of a DOSC. At a given point in time, we must know:

- **How much computation is a DOSC?** That is, with the *fastest processor available*, how much sequential computation can we do in a day?
- **How expensive is that amount of computation?** That is, how much would it cost to produce that much computation at that time, in the *most efficient way possible*?

The first question is rather easy to answer. PassMark[7] gives detailed benchmarking results for a wide variety of current and historical CPUs, including single-threaded-only performance. Given this data, we can then deduce how much computation can the fastest CPU available at a given time do in a day, interpreting the PassMark numbers as an arbitrary unit. (We can do this because the numbers are proportional to sequential performance). This is graphed in Figure 5.5

[7] https://www.cpubenchmark.net/

Figure 5.6: DOSC cost over time

As we would expect, as processors improve, the amount of computation a DOSC corresponds to steadily increases. But given how much computation is a DOSC, how expensive would doing that much computation be? This is a much trickier question.

We first formulate a way of estimating how expensive it is to run a given processor for a day. This has two components:

- **Depreciation**: the value of a processor drops over time. Thus, occupying the processor for a day inherently incurs a depreciation cost. In particular, we *very roughly* estimate depreciation as a uniform 20%/year since the CPU was released. That is, after `k` years, a processor originally priced as `d` dollars would cost `0.8^k * d` dollars.

  We then estimate the depreciation cost of running a particular processor at a particular day as its estimated price on that day, minus its estimated price the next day.

  The hard part here is finding how much CPUs actually cost historically. PassMark lacks historical prices for most CPUs, instead only giving the *latest* price, which is usually much lower for older CPUs. Instead, we use Intel ARK to find the release price of historical CPUs. This does limit us to Intel CPUs, which we do not believe to be a big problem because generally Intel CPUs are more efficient at sequential computation.

- **Energy**: the cost of the electricity required to run the processor for a day. This is estimated using the TDP figures in the PassMark data, combined with electricity prices from FRED. [8]

We can then combine this information with the speed of the processor to deduce how expensive it is to run a given processor until it finishes 1 DOSC worth of work.

This now gets us what we want. At any given point in time, we search through *all previously released processors* to find the most efficient way of computing a DOSC. That cost is then the "DOSC cost", plotted in Figure 5.6. We see that the price is remarkably stable over time, bouncing up and down around 30 US cents. This shows that DOSC is indeed a reasonable peg target.

5.5.3 Stochastic market simulation

We have shown that "on average" Melmint should give a very robust peg, but how would it behave in a wide variety of economic conditions? We build a stochastic simulation of a cryptocurrency market to investigate Melmint's behavior.

Setup We simulate the Melmint mechanism on a simple market model containing the following four variables varying with time:

- Total *sym supply* T in circulation
- Total *mel supply* M in circulation
- Current *sym price* p in DOSCs
- Current *mel price* q in DOSCs

We then simulate fluctuating demand for both syms and mels: every simulated day, both the current sym price and the quantity of mels demanded randomly change by at most 1%. Random adjustment of the sym price is done by simply changing p, while we run the Melmint mechanism to create or destroy mels until the quantity demanded is sym.

Running the Melmint mechanism also changes the sym supply and therefore price; this is harder to model since it depends on the demand curve of syms. As a simplifying assumption, every time the mechanism changes the amount of syms by a factor x, we change the sym price by a factor $x^{-1.1}$: $T \leftarrow Tx \implies p \leftarrow px^{-1.1}$. Thus, decreasing sym supply increases the sym price but also

[8] https://fred.stlouisfed.org/series/APU000072610

the sym market capitalization due to market expectations of further contraction, while increasing sym supply does the opposite. We chose a small exponent of x so that indefinite sym inflation will deplete our implicit reserves — keeping sym market capitalization constant would instead allow an infinite amount of value can be raised from inflation, since $\sum_{x=0}^{\infty} 1/x$ does not converge.

Finally, when the implicit reserve ratio Mq/Tp falls below 1, demand for mels rapidly vanishes and we attempt to destroy 5% of all outstanding mels every day. This simulates a panic where the implicit reserves backing mels fail, and a run on mels happens.

Normal case simulation We run a 5000-day simulation of Melmint, with an initial state of $T = M = 1000$, $q = 1$. We vary our starting sym price p to simulate different "safety margins"; for each p we run the simulation 100 times to determine average behavior. The results are summarized in Figure 5.4.

We note that with $p = 0.5$, almost half of the simulation is spent with a broken peg. This is of course expected since the implicit backing is only 50% of the necessary value. We also note that the sym depreciates to less than half of its original value due to the large amount of sym inflation triggered by attempts to maintain the peg. Qualitatively we see in Figure 5.4a that most of the dips in price come from the $p = 0.5$ case.

As p increases above 1, the peg becomes robust. Only a small fraction of the time is spent with a broken peg, [9] and even during huge devaluations the mel price is very close to the peg. Finally, at $p = 2$, a value that gives an implicit reserve ratio close to our previous predictions, the peg can be considered always solid. We do not see any sign of depegging in Figure 5.4a associated with $p = 2$.

5.5.4 Cryptoeconomic security

We analyzed the stability of Melmint under the assumption that all the mechanisms work without disruption. Now, let us examine what would happen when adversaries deliberately attempt to destabilize the system to prevent the mechanisms from operating.

It would be almost impossible to rigorously deal with *all* possible attacks of this sort — many attacks such as "physically disrupt the processor supply chain to increase DOSC volatility" are impractical to model. Instead, we focus on two particular kinds of attacks common against DeFi mechanisms: *mechanism-level* attacks which attempt to manipulate a Melmint mechanism working on a perfectly trustworthy blockchain, and *blockchain-level* attacks that subvert Melmint by disrupting

[9] Note that our model already simulates a total "bank run" when the backing becomes insufficient, so the measured fraction of time spent with a broken peg takes into account that panic-selling tends to quickly spread to all participants.

the underlying blockchain. In both cases, we assume that the purpose of the attacker is to de-peg the mel from the DOSC and destroy its value stability.

Mechanism-level attacks In a mechanism-level attack, an adversary attempts to disrupt the price of a mel by engaging in actions allowed by the protocol. One possible avenue is simply attacking the peg directly. The attacker might try to sell or buy huge amounts of mels, affecting their price. However, this is extremely costly and ineffective, since the minting mechanism in Section 5.4.3 ensures the peg would stay safe and the attacker would lose a great deal of money. For example, if the attacker tries to buy up mels to increase its price, anybody can repeatedly turn in the DOSC-equivalent of syms to get mels and sell them to the attacker for a profit, pushing the mel price back down. In fact, the robustness of Melmint's mechanism rests on this sort of arbitrage.

Two crucial parts of Melmint, though, do not rest on such an obvious two-way arbitrage: erg minting and the Melswap market. Attacks against minting are not possible without breaking the Elasticoin mechanisms [DB19]. Against the Melswap market, the attacker may want to cause either an overvalued or undervalued sym price to be measured. This would destabilize the system and cause fluctuations in mel prices.

The Melswap market uses a Uniswap-like *constant-product* automated market maker, a commonly used market model with very well-analyzed robustness [Ang+19]. With sufficient liquidity, market manipulation is very difficult. More importantly, because Melmint's "strength" is scaled to existing liquidity, it is very unlikely for it to overreact disastrously in illiquid markets, as in the TerraUSD collapse. [Cha22]

Blockchain-level attacks The attacker may subvert the underlying blockchain in order to attack Melmint. Here we do not consider attacks that totally break the blockchain (say, 51% attacks that cause double spending) as defenses against them should be handled by cryptoeconomic mechanisms within the blockchain itself. Instead, we consider "pathological" strategies that consensus participants (miners in Ethereum, stakeholders in Themelio) may follow that preserve consensus but might damage the stability of Melmint.

In particular, we consider *frontrunning*, where the attacker has knowledge of blockchain transactions in advance of others, and *censorship*, where the attacker prevents transactions from reaching the blockchain. In both cases, we assume an extremely powerful attacker: one that has a consensus monopoly able to control what goes onto the blockchain, rationally seeks to maximize profits, and does not have external incentives such as bribes or threats. Under these circumstances, we do not want to create an incentive for this attacker to do any action that would damage Melmint's stability — thus, the cryptoeconomic incentive structure of the underlying blockchain would suffice.

Table 5.2: Comparison of Melmint to other systems

	Trusted parties	Strong peg?	Reserves
Tether	Issuer	Yes	Bank deposits
MakerDAO	Oracles	Probably	On-chain collateral
Seign. Shares	Oracles	Uncertain	Seigniorage-based
Basis	Oracles	Uncertain	Seigniorage-based
Elasticoin	None	No peg	None
Melmint	None	Yes	Fee-based

In a frontrunning attack, the adversary uses its privileged position to observe Melmint transactions, such as DOSC-minting and sym auctions, before others see them. It is likely possible to extract some profit from this information — for example, by dumping mels just in advance of an emergency devaluation. But in line with existing economic research [DM98; HLS18], we expect any frontrunning to actually increase the efficiency of the markets established by Melmint, helping rather than harming its stability.

Censorship is a much more serious issue. It is quite obvious that with total control over what transactions go onto the blockchain, an adversary can destabilize Melmint as much as she wants — after all, Melmint operates entirely with blockchain-based inputs. However, will there be a mechanism-internal incentive for a blockchain-controller adversary to do so?

Although we have not yet done a rigorous game-theoretical analysis, we conjecture that the answer is no, there's nothing to gain out of manipulating Melmint for a blockchain-controlling adversary. In Themelio's instantiation of Melmint where syms are controlling shares in both protocol revenue and consensus, there is a fairly intuitive argument that manipulation will not be profitable — manipulating the exchange rate would almost certainly cause syms to depreciate, and blockchain-controlling adversaries necessarily own a vast amount of syms. One might guess that large sym-holders would want to censor all sym minting to prevent their share from diluting, but in fact they lack an incentive to do so assuming an efficient market, since any sym minting blocked would only accumulate and happen all at once at the end of the attack, and expectations of this would depreciate syms just as much as actual sym minting would.

5.5.5 Comparison to existing systems

Finally, we compare Melmint with existing work on reducing cryptocurrency volatility. Table 5.2 compares three important aspects of a cryptocurrency issuance mechanism: the *trusted parties*,

whether or not a *strong peg* to some stable index is achieved, and the implicit or explicit *reserves* backing the peg.

We give Tether [Tet] as an example of a traditional centralized stablecoin: it is issued by a trusted party, backed by fiat reserves, and maintains a robust peg assuming the issuer is reliable. MakerDAO [Tea17] is the most popular stablecoin without a trusted issuer, relying on only a trusted oracle that publishes up-to-date exchange rates with the US dollar. It has a unique system roughly reminiscent of non-deliverable forwards used to trade non-convertible currencies in traditional finance; its reserves used to support the coin's value is an on-chain reserve of cryptocurrency almost always worth more than the issued coins.

Seigniorage Shares [Sam19] and its now-defunct [Al-18] descendant Basis [ACD17] are the stablecoins closest in design to Melmint. Like Melmint, both couple the stablecoin with a secondary volatile asset ("shares" or "basebonds" or "syms") that is inflated and deflated to support the stablecoin's peg. Reserves are therefore implicit and roughly equivalent to the market capitalization of the secondary asset. Unfortunately, both seigniorage shares and Basis use secondary assets with value derived solely from expected future inflation (seigniorage) of the main coin. This makes the implicit reserves only sound when there is steadily and rapidly increasing demand; seigniorage in a normal fiat currency is usually a minuscule fraction of total circulating currency. In fact, when demand is expected to decline in the future, the secondary assets would actually have *negative* value! Thus, seigniorage-backed stablecoins may be fundamentally unsound, as several analyses of Basis have concluded.

Finally, Elasticoin [DB19] introduced the concept of a low-volatility trustless cryptocurrency, but it does not have a stable peg and has problems with volatility during periods of low demand. Melmint synthesizes Elasticoin with a Seigniorage Shares-like implicit reserve with a *fee-based* value that's tied to the total economic value of the system, rather than a self-referential monetary policy variable like seigniorage. Thus, Melmint achieves the "holy grail": no trusted parties, a strong peg, and robust reserves.

5.6 Conclusion

In this section, we presented Melmint, a new cryptocurrency issuance scheme that robustly pegs issued coins to the DOSC, a unit measuring the cost of sequential computation for a day. This is done by combining Elasticoin, an existing algorithm for measuring the value of a DOSC, with a strong implicit reserve based on diluting shares of fee revenue. Unlike all other stablecoin proposals, Melmint operates without any trusted issuers or oracles while maintaining a robust peg. We show through both qualitative argument and extensive stochastic simulations that Melmint does indeed achieve its goals.

Chapter 6

Astrape: simple private payment channels

In its pursuit of strong endogenous trust, Themelio chose a paradigm different from conventional "platform blockchains" like Ethereum. Application development patterns suitable for platform blockchains often do not port over in a straightforward manner, since Themelio does not provide a convenient substrate on which apps can be easily created directly.

Instead, applications in Themelio would most likely interface with *mid-level protocols*, including both usual "layer-2" protocols like payment channels and endogenously trustworthy implementations of conventional secure systems like PKIs. These protocols are not part of the consensus-critical core of the blockchain, allowing them to evolve over time without compromising Themelio's endogenous trust.

As our first example, we present Astrape, a novel anonymous payment channel protocol. Astrape was originally formulated as a blockchain-neutral payment channel network that can run on blockchains ranging from Bitcoin Cash to Ethereum, so as a payment channel network itself, it is not exactly Themelio-specific.

But in Themelio, "layer-2" protocols can be used to extend blockchain-backed security to non-blockchain applications much more readily, since embeddable light clients allow applications to be able to interface with the on-chain portion of these protocols without problematic trust assumptions. This gives much more significance to these mid-level protocols than they would have if they were to be built on most other blockchains.

We will return to exactly how light clients allow new ways of composing these mid-level protocols in Chapter 8.

6.1 Introduction

6.1.1 Payment channel networks

Blockchain cryptocurrencies are gaining in popularity and becoming a significant alternative to traditional government-issued money. For instance, over 300,000 Bitcoin transactions alone [Blo21] are processed every day. Unfortunately, such high demand inevitably leads to well-known scalability barriers [Cro+16]. Bitcoin, for instance, processes less than 10 transactions every second [McC+16], far less than a reasonable global payment system.

Payment channels [DW15] are a common technique to scale cryptocurrency transactions. In a nutshell, Alice and Bob open a payment channel by submitting a single transaction to the blockchain, locking up a sum of cryptocurrency from both parties. They can then pay each other by simply mutually signing a division of the locked money. Additional blockchain transactions are required only when the channel is closed by submitting an up-to-date signed division, unlocking the latest balances of Alice and Bob. This allows most activity to remain off-chain, while retaining the blockchain for final settlement: as long as the blockchain is secure, nobody can steal funds. More importantly, payment channels can be organized into *payment-channel networks* (PCNs) [McC+16], where users without any open channels between them can pay each other through intermediaries.

6.1.2 Anonymity in PCNs

Unfortunately, "first-generation" PCNs based on the HTLC (hash time-locked contract), such as Lightning Network [DW15], have a significant problem — poor anonymity [Mal+17]. In the worst case, HTLC payments are as transparently linkable as blockchain payments [Mal+17], threatening the improved privacy that is often cited [Van16; You+21] as a benefit of PCNs. Furthermore, naive implementations fall victim to subtle fee-stealing attacks, like the "wormhole attack" [Mal+19], that threaten economic viability.

A sizable body of existing work on fixing PCN security and privacy exists. On one hand, specialized constructions achieve strong anonymity in specific settings, such as Bolt [GM17] for hub-based PCNs on the Zcash blockchain, providing for indistinguishability of two concurrent transactions even when all intermediaries are malicious. On the other hand, general solutions for all PCN topologies, like Fulgor [Mal+17] and the AMHL (Anonymous Multi-hop Locks) family [Mal+19], achieve a somewhat weaker, topology-dependent notion of anonymity: *relationship anonymity* [Bac+13; Lai+17]. This property, common to onion-routing and other anonymous communication protocols, means that two concurrent transactions cannot be distinguished as long as they share at least one honest intermediary.

6.1.3 Cryptographic constructions in PCNs

Unfortunately, there remains a shortcoming common to all existing anonymous PCN constructions — custom, often number-theoretic and sometimes complex cryptographic primitives. No existing anonymous PCN construction limits itself to the bare-bones cryptographic primitives used in HTLC — black-box access to a generic signature scheme and hash function. For example, AMHL uses either homomorphic one-way functions or special constructions that exploit the mathematical structure of ECDSA or Schnorr signatures and Tumblebit uses a custom cryptosystem based on the RSA assumption. Blitz [AMM21], though relying on an ostensibly black-box signature scheme, requires it to have a property[1] that rules out many post-quantum signature schemes.

The key objective of these constructions is reconciling anonymity with *balance security* — ensuring that the entire multi-hop transaction either completes correctly or reverts all money to the sender. The former requires hiding important information, while the second requires verifying it, making a zero-knowledge protocol a seemingly natural fit. Indeed, for hub-based anonymous PCNs, a strong zero-knowledge cryptosystem is likely necessary — privacy against an adversary controlling *everyone* other than the counterparties is a difficult goal to achieve.

However, it is unclear that relationship anonymity requires sophisticated techniques. Relationship anonymity appears to be relatively "easy" elsewhere. Well-understood anonymous constructs like onion routing and mix networks exist for communication with no more than standard primitives used in secure communication (symmetric and asymmetric encryption). Of course, communication is probably easier — indeed some go beyond relationship anonymity with only simple cryptography — but it seems plausible that PCNs can use similarly elementary primitives to achieve anonymity.

Furthermore, "boring" cryptography has practical advantages. For one, non-standard cryptography poses significant barriers to adoption. Reliable and performant implementations of novel cryptographic functions are difficult to obtain, and tight coupling between a PCN protocol and a particular cryptographic construction makes swapping out primitives impossible. With use of black-box cryptography, a system is *generic over cryptographic hardness assumptions*. Instead of assuming specific problems like the RSA or discrete-log problems are hard, we only need to assume that there exists, for instance, *some* secure signature scheme and *some* secure hash function.

Thus, we believe that efficient yet privacy-preserving PCNs that only use well-understood and easily replaced black-box cryptographic primitives are crucial to usable PCNs. In fact, AMHL's authors already proposed that "an interesting question related to [anonymous PCN constructions] is under which class of hard problems such a primitive exists" [Mal+19] that they conjecturally answered with linear homomorphic one-way functions.

[1] In particular, the ability for any party, given any public key, to generate new public keys that correspond to the same private key yet are unlinkable to the previous public key. This is crucial to the "stealth addresses" that Blitz's pseudonymous privacy rests upon.

6.1.4 Our contributions

We present Astrape,[2] a PCN protocol that limits itself to "boring", generic cryptography already used in HTLC, yet achieves strong relationship anonymity. Despite achieving comparable security, privacy, and performance to other anonymous PCN constructions, Astrape does not introduce any cryptographic constructs other than those used in HTLC. This is accomplished using a novel construct reminiscent of onion routing that avoids the use of any form of zero-knowledge verification.

- Section 6.2 discusses existing payment channel networks, focusing on their security and privacy properties. We show that first-generation PCNs have significant privacy and security problems, and discuss existing work on improving them.
- Section 6.3 introduces a formal model of PCN constructions, "generalized multi-hop locks" (GMHL), by generalizing "anonymous multi-hop locks" [Mal+19]. We model the security and privacy properties we desire within this model.
- Within the GMHL framework, we introduce Astrape, a new PCN construction that solves the security and privacy issues of HTLC-based constructs with only black-box cryptographic primitives in Section 6.4. Strong privacy without new cryptographic primitives solves an open problem in the field. Astrape builds upon existing work [Mal+17] on private PCNs but uses a new technique that, at a modest cost in lock and unlock size as well as script complexity, avoids the need for verifying hidden information.
- Section 6.5 presents a security and privacy analysis, showing that Astrape achieves the same security and privacy goals as existing work like Fulgor [Mal+17] and AMHL [Mal+19]. Section 6.6 discusses some potential difficulties in practical deployment of Astrape — in particular, difficulty in deployment to blockchains without smart contracts — and presents solutions.
- Finally, in Section 6.7 we implement Astrape and show that on average, Astrape requires less computation or communication than state-of-the-art private PCN constructions. We show that Astrape can easily be ported to different blockchains.

[2] Greek for "lightning", pronounced "As-trah-pee".

6.2 Background and related work

6.2.1 Payment channels

The fundamental building block of payment channel networks is the bilateral payment channel [McC+16]. A payment channel is a construct where two parties Alice and Bob deposit money into a blockchain-enforced "vault" that can be only opened with signatures from both Alice and Bob. Alice and Bob can then send each other money by privately agreeing on the distribution of funds between them by signing dated statements. When one of the parties needs to access the funds, they can simply open the vault using any statement that was produced in the course of using the payment channel, subject to a short period of time where the other party may override it with a later-dated statement. This mechanism ensures security of funds even if one of the parties is malicious.

6.2.2 First-generation PCNs with HTLC

An extremely useful property of payment channels is that they can be used to construct *payment channel networks* (PCNs) [McC+16; DRO18; Cro+16], allowing users without channels directly between each other to pay each other via intermediaries. At the heart of any PCN is a *secure multi-hop transaction* mechanism — some way of Alice paying Bob to pay Carol without any trust in Bob. Most PCNs implement this using a smart contract known as the *Hash Time-Lock Contract* (HTLC). An HTLC is parameterized over a *sender* Alice, the *recipient* Bob, a deadline t, and a *puzzle* s. It locks up a certain amount of money, unlocking it according to the following rules:

- The money goes to Bob if he produces π where $H(\pi) = s$ before time t, where H is a secure hash function.
- Otherwise, the money goes to Alice.

We can use HTLC to construct secure multi-hop transactions. Consider a sender U_0 wishing to send money to a recipient U_n through untrusted intermediaries $U_1, \ldots, U_{n-1}$. At first, U_0 will generate a random π and $s = H(\pi)$, while sending the pair (π, s) to U_n over a secure channel. U_0 can then lock money in a HTLC parameterized over U_0, U_1, s, t_1, notifying U_1. U_1 would send an HTLC over U_1, U_2, s, t_2, notifying U_2, and so on. The deadline must become earlier at each step — $t_1 > t_2 > \cdots > t_n$ — this ensures that in case of an uncooperative or malicious intermediary, funds always revert to the sender.

The payment eventually will be routed to U_n, who will receive an HTLC over U_{n-1}, U_n, s, t_n. The recipient will claim the money by providing π; this allows U_{n-1} to claim money from U_{n-2}

using the same π, and so on, until all outstanding HTLC contracts are fulfilled. U_0 has successfully sent money to U_n, while the preimage resistance of H prevents any intermediary from stealing the funds.

6.2.3 Hub-based anonymous payment channels

Unfortunately, HTLC has an inherent privacy problem — a common identifier $s = H(\pi)$ visible to all nodes in the payment path [GM17; Mal+17; Mal+19]. This motivates *anonymous* PCN design. *Hub-based* approaches form the earliest kind of anonymous PCN design. Here, the shape of the network is limited to a star topology with users communicating with a centralized hub. Some solutions are highly specialized, such as Green and Miers' Bolt [GM17], which relies on the Zcash blockchain's zero-knowledge cryptography. Other solutions, such as Tumblebit [Hei+17] and the more recent A2L [TMM21], provide more general solutions that work on a wide variety of blockchains.

Hub-based PCN constructions tackle the difficult problem of providing unlinkability between transactions despite the existence of only a single untrusted intermediary. It is therefore unsurprising that specialized cryptography is needed to protect anonymity. On the other hand, observations of real-world PCNs like the Lightning Network, as well as economic analysis [Eng+17], show that actual PCNs often have intricate topologies without dominating hubs. General, topology-agnostic solutions are thus more important to deploying private PCNs in practice.

6.2.4 Relationship-anonymous payment channels

Unlike hub-based approaches, where no intermediaries are trusted, general private PCN constructions target *relationship anonymity*. This concept, shared with onion routing and other anonymous communication protocols, assumes at least one honest intermediary. Thus intermediaries are in fact crucial to relationship-anonymous PCNs' privacy properties. Like most hub-based approaches, relationship-anonymous payment channels do not by themselves deal with information leaked by side channels such as timing and value. We return to this subject in Section 6.6.2.

The earliest solution to PCN privacy in this family was probably Fulgor and Rayo [Mal+17], a closely related pair of constructions that can be ported to almost all HTLC-based PCNs. Fulgor/Rayo combines a "multi-hop HTLC" contract with out-of-band ZKPs to remove the common identifier across payment hops.

In a later work, Malavolta et al. [Mal+19] introduced *anonymous multi-hop locks* (AMHL), a rigorous theoretical framework for analyzing private PCN contracts. The AMHL paper provided a concrete instantiation using linear homomorphic one-way functions (hOWFs), as well as a conjecture

that hOWFs are necessary for implementing anonymous PCNs. They also presented a variant that uses a clever encoding of homomorphic encryption in ECDSA to be used in ECDSA-based cryptocurrencies like Bitcoin. The latter "scriptless" variant was generalized in later work to a notion of adaptor signatures [Aum+21], where a signature scheme like ECDSA is "mangled" in such a way that a correct signature reveals a secret based on a cryptographic condition. The authors of AMHL also discovered "wormhole attacks" on HTLC-based PCNs. These attacks exploit a fundamental flaw in the HTLC construction to allow malicious intermediaries to steal transaction fees from honest ones, a problem that AMHL's anonymity techniques also solve.

More recently, Blitz [AMM21] introduced *one-phase* payment channels that support multi-hop payments without a two-phase separation of coin creation and spending, improving performance and reliability. Blitz also achieves stronger anonymity than HTLC, but its notion of anonymity is strictly weaker than the relationship anonymity of AMHL and Fulgor/Rayo. Other relationship-anonymous systems consider powerful adversaries that control most nodes and achieve indistinguishability of concurrent transactions, but Blitz considers local adversaries controlling a single intermediary and limits itself to hiding the rest of the path from this intermediary.

6.3 Our approach

As we argued in Section 6.1.3, all of these existing solutions share an undesirable reliance on either custom cryptographic constructions or primitives with special properties, like Blitz's stealth-address signature schemes. This causes inflexibility, difficult implementation, and an inability to respond to cryptanalytic breakthroughs like practical quantum computing.

Astrape is our solution to this problem. We show with a novel design that avoids the zero-knowledge verification paradigm, anonymous and atomic multi-hop transactions can be constructed with nothing but the two building blocks of HTLCs — hashing and signatures. Unlike existing work, no specific assumptions about the structure of the hash function or signature scheme are made, allowing Astrape to be easily ported to different concrete cryptographic primitives and its security properties to "fall out" from those of the primitives. This also allows Astrape to achieve high performance on commodity hardware using standard cryptographic libraries.

6.3.1 Generalized multi-hop locks

In our discussion of Astrape, we avoid describing the concrete details of a specific payment channel network and cryptocurrency. Instead, we introduce an abstract model — generalized multi-hop locks. This model readily generalizes to different families of payment channel networks.

We model a *sender*, U_0, sending money to a *receiver*, U_n, through intermediaries $U_1, \ldots, U_{n-1}$. We assume a "source routing" model, where the graph of all valid payment paths in the network is publicly known and the sender can choose any valid path to the recipient. After an *initialization* phase where the sender may securely communicate parameters to each hop, each user U_i where $i < n$ *creates* a *coin* and notifies U_{i+1}. This coin is simply a contract ℓ_{i+1} known as a *lock script*, that essentially releases money to U_{i+1} given a certain key k_{i+1}. We call this lock the *right lock* of U_i and the *left lock* of U_{i+1}.

Finally, the payment completes once all coins created in the protocol have been unlocked and spent by fulfilling their lock scripts. Typically, this happens through a chain reaction where the recipient's left lock ℓ_n is unlocked, allowing U_{n-1} to unlock its left lock, etc.

Formally, we model a GMHL over a set of participants U_i as a tuple of four PPT algorithms $\mathbb{L} = (\mathsf{Init}, \mathsf{Create}, \mathsf{Unlock}, \mathsf{Vf})$, defined as follows:

Definition 3: A GMHL $\mathbb{L} = (\mathsf{Init}, \mathsf{Create}, \mathsf{Unlock}, \mathsf{Vf})$

consists of the following polynomial-time protocols:

1. $\langle s_0^I, \ldots, (s_n^I, k_n)) \rangle \Leftarrow \langle \mathsf{Init}_{U_0}(1^\lambda, U_1, \ldots, U_n), \mathsf{Init}_{U_1}, \ldots, \mathsf{Init}_{U_n} \rangle$: the initialization protocol, started by the sender U_0, that takes in a security parameter 1^λ and the identities of all hops U_i and returns an initial state s_i^I to all users U_i. Additionally, the recipient receives a key k_n.

2. $\langle (\ell_i, s_{i-1}^R), (\ell_i, s_i^L) \rangle \Leftarrow \langle \mathsf{Create}_{U_{i-1}}(s_{i-1}^I), \mathsf{Create}_{U_i}(s_i^I) \rangle$: the coin-creating protocol run between two adjacent hops U_{i-1} and U_i, creating the "coin sent from U_{i-1} to U_i". This includes a lock representation ℓ_i as well as additional state on both ends — unlocking the lock represented by ℓ_i releases the money.

3. $k_i \Leftarrow \mathsf{Unlock}_{U_i}(\ell_{i+1}, (s_i^I, s_i^L, s_i^R), k_{i+1})$: the coin-spending protocol, run by each intermediary U_i where $i < n$, obtains a valid unlocking key k_i for the "left lock" ℓ_i given its "right lock" ℓ_{i+1}, its unlocking key k_{i+1} (already verified by Vf below), and U_i's internal state.

4. $\{0, 1\} \Leftarrow \mathsf{Vf}(\ell, k)$: given a lock representation ℓ and an unlocking key k, return 1 iff the k is a valid solution to the lock ℓ

Generalizability to non-PCN systems. We note here that GMHL makes no mention of typical PCN components such as channels, the blockchain, etc. This is because GMHL is actually agnostic of *how* exactly the locks are evaluated and enforced. In a typical PCN, these locks will be executed within bilateral payment channels, falling back to a public blockchain for final settlement.

However, other enforcement mechanisms can be used. Notably, all the locks could simply be contracts directly executing on a blockchain. In this way, any anonymous PCN formulated in the

GMHL model is equivalent to a specification for a provably anonymous *on-chain, multi-hop coin tumbling service* that can anonymize entirely on-chain payments by routing them through multiple intermediaries.

Comparison to existing work. GMHL is an extension of *anonymous multi-hop locks*, the model used in the eponymous paper by Malavolta et al. [Mal+19]. In particular, AMHL defines an anonymous PCN construction in terms of the operations KGen, Setup, Lock, Rel, Vf, four of which correspond to GMHL functions.

Although AMHL's model is useful, we could not use it verbatim. This is largely because AMHL's original definition [Mal+19] also included its security and privacy properties, while we wish to be able to use the same framework in a purely *syntactic* fashion to discuss PCNs with other security and anonymity goals.

Nevertheless, GMHL can be considered as AMHL, reworded and used in a more general context. As we will soon see, Astrape's desired security and privacy properties are actually very similar to those of AMHL, though we will consider other systems formulated in the GMHL framework along the way. Astrape can be considered an alternative implementation of the same "anonymous multi-hop locks" [Mal+19] construct.

6.3.2 Security and execution model

Now that we have a model to discuss PCN constructions, we can discuss our security model, as well as a model of the GMHL execution environment in which Astrape will execute.

Active adversary. We use a similar adversary model to that of AMHL [Mal+19]. That is, we model an adversary $\mathcal{A}$ with access to a functionality $\mathsf{corrupt}(U_i)$ that takes in the identifier of any user U_i and provides the attacker with the complete internal state of U_i. The adversary will also see all incoming and outgoing communication of U_i. $\mathsf{corrupt}(U_i)$ will also give the adversary active control of U_i, allowing it to impersonate U_i when communicating with other participants.

Anonymous communication. We assume there is a secure and anonymous message transmission functionality $\mathcal{F}_{\text{anon}}$ that allows any participant to send messages to any other participant. Messages sent by an honest (non-corrupted) user with $\mathcal{F}_{\text{anon}}$ hide the identity of the sender and cannot be read by the adversary, although the adversary may arbitrarily delay messages.

There are many ways of implementing $\mathcal{F}_{\text{anon}}$, the exact choice of which is outside the scope of this paper. One solution recommended by existing work [Mal+17; Mal+19] is an onion-routing

circuit constructed over the same set of users U_i, constructed with a provably private protocol like Sphinx [DG09]. Public networks such as Tor may also be used to implement $\mathcal{F}_{\text{anon}}$.

Exposed lock activity. In contrast to communication, *lock activity* — the content of all locks being created, as well as the unlocking keys during unlocking — is not secure. This is because in practice, lock activity often happens on public media like blockchains. We pessimistically assume that the adversary can see all lock activity, while a non-adversary only sees lock activity concerning locks that it sends and receives.

Liveness and timeouts. We assume that every coin lock ℓ_i comes with an appropriate timeout that will return money to U_{i-1} (i.e., able to be unlocked by a signature from U_{i-1} after the timeout) if U_i does not take action. We also assume that each left lock ℓ_i's deadline is at least δ later than that of the right lock ℓ_{i+1}, where δ is an upper bound on network latency between honest parties, even under disruption by the adversary. In the most common setting of a PCN consisting of bilateral payment channels backed by a blockchain, this is essentially a blockchain censorship-resistance assumption. With a liveness assumption, we can then omit timeout handling from the description of the protocol, in line with related work (such as AMHL [Mal+19]).

Infallible lock execution. We formulate Astrape in the GMHL model, and assume the existence of a mechanism that will guarantee that cryptocurrency locks are always correctly executed in the face of arbitrary adversarial activity. In practice, both bilateral payment channels falling back to a general-purpose public blockchain (like Ethereum) and direct use of this blockchain are good approximations of this mechanism.

Lock functionality. We assume that inside our on-chain contracts we are able to use at least the following operations:

- *Concatenation*, producing a bitstring $x||y$ of length $n + m$ from two bitstrings x, y, where x has length n and y has length m.
- *Bitwise XOR*, producing a bitstring $x \oplus y$ from two bitstrings x, y

as well as the cryptographic hash function H defined below. This is the case for Mel's scripting language, Melodeon, as well as the languages of essentially all smart-contract blockchains like Ethereum. But one implication of this assumption is that PCNs on blockchains with highly restricted scripting languages, like Bitcoin, cannot use Astrape.

Cryptographic assumptions. One of Astrape's main goals is to make minimal cryptographic assumptions. We assume only:

- *Generic cryptographic hash function*. We assume a hash function H, modeled as a random oracle for the purpose of security proofs, producing λ bits of output, where 1^λ is the security parameter. We use the random oracle both as a pseudorandom function and as a commitment scheme, which is well known [Cam+18] to be secure.
- *Generic signature scheme*. We assume a secure signature scheme that allows for authenticated communication between any two users U_i and U_j.

6.3.3 Security and privacy goals

Against the adversary we described above, we want to achieve the following security and privacy objectives:

Relationship anonymity. Given two simultaneous payments between different senders $S_{\{0,1\}}$ and receivers $R_{\{0,1\}}$ with payment paths of the same length intersecting at the same position at at least one honest intermediate user, an adversary corrupting all of the other intermediate users cannot determine, with probability non-negligibly better than $1/2$ (guessing), whether S_0 paid R_0 and S_1 paid R_1, or S_0 paid R_1 and S_1 paid R_0. This is an established standard for anonymity in payment channels [Mal+17; Mal+19] and is analogous to similar definitions for anonymous communication [PH10; Bac+13]. It is important to note that the adversary is not allowed to corrupt the sender — senders always know who they are sending money to.

Balance security. For an honest user U_i, if its right lock ℓ_i is unlocked, U_i must always be able to unlock its left lock ℓ_{i-1} even if all other users are corrupt. Combined with the timeouts mentioned in our security model, this guarantees that no intermediary node can lose money even if everybody else conspires against it.

Wormhole resistance. We need to be immune to the *wormhole attack* on PCNs, where malicious intermediaries steal fees from other intermediaries. The reason why is rather subtle [Mal+19], but for our purposes this means that given an honest sender and an honest intermediary U_{i+1}, ℓ_i cannot be spent by U_i until U_{i+1} spends ℓ_{i+1}. Intuitively, this prevents honest intermediaries from being "left out".

6.4 Construction

6.4.1 Core idea: balance security + honest-sender anonymity

Unlike existing systems that utilize the mathematical properties of some cryptographic construction to build a secure and anonymous primitive, Astrape is constructed out of two separate *broken* constructions, both of which use boring cryptography and are straightforward to describe:

- **XorCake**, which has relationship anonymity but lacks balance security if the sender U_0 is malicious
- **HashOnion** which has balance security, but *loses relationship anonymity* in the Unlock phase. That is, an adversary limited only to observing Init and Create cannot break relationship anonymity, but an adversary observing Unlock can.

The key insight here is that if we can combine XorCake and HashOnion in such a way to ensure that HashOnion's Unlock phase *can only reveal information when the sender is malicious*, we obtain a system, Astrape, that has both relationship anonymity and balance security. This is because the definition of relationship anonymity assumes an honest sender: if the sender is compromised, it can always simply tell the adversary the identity of its counterparty, breaking anonymity trivially. It is important to note that such a composition *does not in any way weaken anonymity* compared to existing "up-front anonymity" systems like AMHL, even in the most pessimistic case. If the sender is honest, Astrape reduces to XorCake and has strong anonymity. If the sender is compromised, on the other hand, the adversary already knows who is sending money to whom in all relationship-anonymous PCNs, as the sender must know the recipient to initiate the payment. In both cases, Astrape has anonymity equivalent to that of existing systems like AMHL and Fulgor.[3]

We now describe XorCake and HashOnion, and their composition into Astrape.

6.4.2 XorCake: anonymous but insecure against malicious senders

Let us first describe XorCake's construction. XorCake is an extremely simple construction borrowed from "multi-hop HTLC", a building block of Fulgor [Mal+17]. It has relationship anonymity, but not balance security against malicious senders.

[3] In a sense then, Astrape has "pseudo-optimistic" anonymity. Its design superficially suggests an optimistic construction with an anonymous "happy path" and a non-anonymous "unhappy path", but a more careful analysis in Section 6.5.1 reveals that the latter non-anonymity is illusory — the sender can always prevent the "unhappy" path from deanonymizing the transaction even if all other parties are malicious.

Recall that in GMHL, the sender (U_0) wishes to send a sum of money to the recipient (U_n) through $U_1, \ldots, U_{n-1}$. At the beginning of the transaction, the sender samples n independent λ-bit random strings $(r_1, \ldots, r_n)$. Then, for all $i \in 1, \ldots, n$, she sets n values $s_i = H(r_i \oplus r_{i+1} \oplus \cdots \oplus r_n)$, where H is a secure hash function. That is, s_i is simply the hash of the XOR of all the values r_j for $j \geq i$. U_0 then uses the anonymous channel $\mathcal{F}_{\text{anon}}$ to provide U_n the values (r_n, s_n) and all the other U_i with (r_i, s_i, s_{i+1}).

Then, for each pair of neighboring nodes (U_i, U_{i+1}), U_i sends U_{i+1} a coin encumbered by a regular HTLC ℓ_{i+1} asking for the preimage of s_{i+1}. U_n knows how to unlock ℓ_n, and the solution would let U_{n-1} unlock ℓ_{n-1}, and so on. That is, each lock ℓ_i is simply an HTLC contract asking for the preimage of s_i.

XorCake by itself satisfies relationship anonymity. A full proof is available in the Fulgor paper from which XorCake was borrowed [Mal+19], but intuitively this is because r_i will be randomly distributed over the space of possible strings because H behaves like a random oracle. This means that unlike in HTLC, no two nodes U_i and U_j can deduce that they are part of the same payment path unless they are adjacent.

State-mismatch attack. Unfortunately, *XorCake does not have balance security*. Consider a malicious sender who follows the protocol correctly, except for sending an incorrect r_i to U_i. (Note that U_i cannot detect that r_i is incorrect given a secure hash function.) Then, when ℓ_{i+1} is unlocked, ℓ_i cannot be spent! In an actual PCN such as the Lightning Network, all coins "left" of U_i will time out, letting the money go back to U_0. U_0 paid U_n with U_i's money instead of her own. We call this the "state-mismatch attack", and because of it, XorCake is not a viable PCN construction on its own. In Fulgor, XorCake was combined with out-of-band zero-knowledge proofs of the correctness of r_i, but as we will see shortly, Astrape can dispense with them.

6.4.3 HashOnion: secure but eventually non-anonymous

We now present HashOnion, a PCN construction that has balance security but not relationship anonymity. Note that unlike HTLC, HashOnion's non-anonymity stems entirely from information leaked in the Unlock phase, a property we will leverage to build a fully anonymous construction combining HashOnion and XorCake.

At the beginning of the transaction, U_0 generates random values s_i for $i \in \{1, \ldots, n\}$, then "onion-like" values x_i, recursively defined as $x_i = H(s_i || x_{i+1})$, $x_n = H(s_n || 0^\lambda)$.

Essentially, x_i is a value that commits to all s_j where $j \geq i$. An onion-like commitment is used rather than a "flat" commitment (say, a hash of all s_j where $j \geq i$) as it is crucial for balance security, as we will soon see.

For all intermediate nodes $0 < i < n$, the sender sends (x_{i+1}, s_i) to U_i, while for the destination, the sender sends s_n. Then, each intermediary U_{i-1} sends to its successor U_i a lock ℓ_i, which can be only be unlocked by some $k_i = (s_i, \ldots, s_n)$ where $H(s_i||H(s_{i+1}||H(\ldots H(s_n||0^\lambda)))) = x_i$. U_{i-1} constructs this lock from the x_i it received from the sender. Finally, during the unlock phase, the recipient U_n solves ℓ_n with $k_n = (s_n)$. This allows each U_i to spend ℓ_i, completing the transaction.

For balance security, we need to show that with a solution $k_{i+1} = (s_{i+1}, \ldots, s_n)$ to ℓ_{i+1}, and s_i, we can always construct a solution to ℓ_i . This is obvious: we just add s_i to the solution: $k_i = (s_i, s_{i+1}, \ldots, s_n)$.

One subtle problem is that U_i needs to make sure that its left lock is actually the correct ℓ_i and not some bad ℓ_i' parameterized over some $x_i' \neq H(s_i||x_{i+1})$. Otherwise, its right lock might get unlocked with a solution that does not let it unlock its left lock. Fortunately, this is easy: given s_i, x_{i+1} from the sender, U_i can just check that its left lock, parameterized over some x_i, matches $x_i = H(s_i||x_{i+1})$ before sending out ℓ_{i+1} (parameterized with x_{i+1}) to the next hop. Thus, every user can make sure that if its right lock is unlocked, so can its left lock, so balance security holds.

We also see that although the unlocking procedure breaks relationship anonymity by revealing all the s_i, before the unlock happens, HashOnion does have relationship anonymity. This is because the adversary cannot connect the different x_i as long as one s_i remains secret — that of the one honest intermediary.

6.4.4 Securing XorCake+HashOnion

We now move on to composing XorCake and HashOnion. We do so by creating a variant of HashOnion that embeds XorCake and recognizes an *inconsistency witness*. That is, this variant of HashOnion will unlock only when given a combination of values that proves an attempt by the sender to execute a state-mismatch attack for XorCake.

To construct such a lock, after generating the XorCake parameters, U_0 creates n λ-bit values x_i recursively:

$$x_n = o_n, \qquad x_i = H(\overbrace{r_i||s_i||s_{i+1}}^{\text{XorCake parameters}}||o_i||x_{i+1})$$

where o_i is a random nonce sampled uniformly from all possible λ-bit values.[4] The intuition here is that x_i *commits* to all the information U_0 would give to all hops U_j where $j \geq i$.

[4] $||$ denotes concatenation. In our case, it is possible to unambiguously separate concatenated values, since we only ever concatenate λ-bit values.

Afterwards, the sender then uses $\mathcal{F}_{\text{anon}}$ to send (o_i, x_i, x_{i+1}) , in addition to the XorCake parameters (r_i, s_i, s_{i+1}), to every hop i. Every hop U_i checks that all the parameters are consistent with each other.

We next consider what will happen if the sender attempts to fool an intermediate hop U_i with a state-mismatch attack. U_{i+1} would unlock its left lock ℓ_{i+1} by giving k_{i+1} where $H(k_{i+1}) = s_{i+1}$ but $H(r_i \oplus k_{i+1}) \neq s_i$. This then causes U_i to fail to unlock its left lock.

But this attempt allows U_i to generate a cryptographic witness verifiable to anybody knowing x_i: λ-bit values k_{i+1}, r_i, s_i, s_{i+1}, o_i, x_{i+1} where:

$$H(k_{i+1}) = s_{i+1}, H(r_i \oplus k_{i+1}) \neq s_i, H(r_i||s_i||s_{i+1}||o_i||x_{i+1}) = x_i$$

This inconsistency witness proves that the preimage of s_{i+1} XOR-ed with r_i does not equal the preimage of s_i, demonstrating that the values given to U_i are inconsistent and that U_0 is corrupt. Since U_{i-1} knows x_i, U_i can therefore prove that it was a victim of a state-mismatch attack to U_{i-1}.

Since x_i commits to *all* XorCake initialization states "rightwards" of U_i, U_i, in cooperation with U_{i-1}, can also produce a witness that U_{i-2} can verify using x_{i-1}. This is simply a set of λ-bit values k_{i+1}, r_{i-1}, s_{i-1}, r_i, s_i,s_{i+1}, x_i, o_i, o_{i-1}, x_{i+1} where:

$$H(k_{i+1}) = s_{i+1}, H(r_i \oplus k_{i+1}) \neq s_i,$$

$$H(r_i||s_i||s_{i+1}||o_i||x_{i+1}) = x_i, H(r_{i-1}||s_{i-1}||s_i||o_{i-1}||x_i) = x_{i-1}$$

We can clearly extend this idea all the way back to U_1 — given a witness demonstrating a state-mismatch attack against U_i, U_{i-1} can verify the witness and generate a similar one verifiable by U_{i-2}, and so on. This forms the core construction that Astrape uses to fix XorCake's lack of balance security.

6.4.5 Complete construction

We now present the complete construction of Astrape, as formalized in Figure 6.1 within the GMHL framework. Note that we use the notation $\mathsf{Tag}[x_1, \ldots, x_n]$ to represent a n-tuple of values with an arbitrary "tag" that identifies the type of value.

Initialization. In the first phase, represented as Init in GMHL, the sender U_0 first establishes communication to the n hops $U_1, \ldots, U_n$, the last one of which is the receiver. When talking to intermediaries and the recipient, U_0 uses our abstract functionality $\mathcal{F}_{\text{anon}}$.

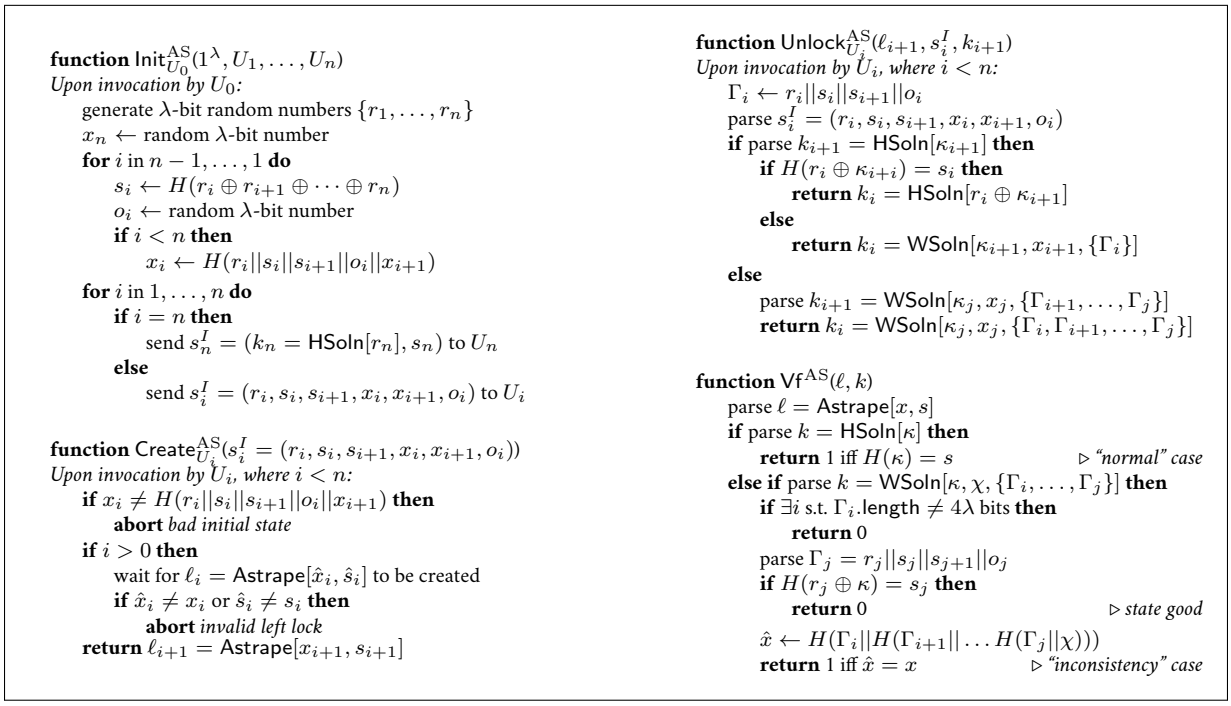


Figure 6.1: Astrape as a GMHL protocol

The sender then generates random λ-bit strings $(r_1, \ldots, r_n)$ and $(o_1, \ldots, o_n)$, deriving $s_i = H(r_i \oplus r_{i+1} \oplus \cdots \oplus r_n)$ and $x_i = H(r_i||s_i||s_{i+1}||o_i||x_{i+1})$; $x_n = H(o_n)$. She then sends to each intermediate hop U_i the tuple $s_i^I = (r_i, s_i, s_{i+1}, x_i, x_{i+1}, o_i)$ and gives the last hop U_n the initial state $s_n^I = (r_n, s_n)$ and the unlocking key $k_n = \mathsf{HSoln}[r_n]$.

Creating the coins. We now move on to Create, where all the coins are initially locked. U_0 then sends U_1 a coin encumbered with a lock represented as $\ell_1 = \mathsf{Astrape}[x_1, s_1]$. When each hop U_i receives a correctly formatted coin from its previous hop U_{i-1}, it sends the next hop U_{i+1} a coin with a lock $\ell_{i+1} = \mathsf{Astrape}[x_{i+1}, s_{i+1}]$. Note that U_i checks whether its left lock is consistent with the parameters it received from U_0; this ensures that when U_i's right lock unlocks later, U_i can always construct a solution for its left lock. If the checks fail, Create aborts, and all of the locks will eventually time out (see Section 6.3.2), returning money to the sender.

As specified in Vf, each of these locks ℓ_i can be spent either through solving a XorCake-type puzzle to find the preimage of s_i (the "normal" case) or by presenting an inconsistency witness with a HashOnion-type witness demonstrating x_i's commitment to inconsistent data (the "inconsistency"

case). After all the transactions with Astrape-encumbered coins are sent, U_n can finally claim its money, triggering the next phase of the protocol.

Unlocking the coins. The last step is Unlock. After receiving the final coin from U_{n-1}, the recipient unlocks its lock ℓ_n by providing to Vf the preimage of the HTLC puzzle: $k_n = \mathsf{HSoln}[r_n]$ — this is the only way an honest recipient can claim the money in a payment originating from an honest sender. Each intermediate node U_i reacts when its right lock ℓ_{i+1} is unlocked with key k_{i+1}:

- If U_{i+1} solved the HTLC puzzle with $k_{i+1} = \mathsf{HSoln}[\kappa_{i+1}]$, construct $\kappa_i = r_i \oplus \kappa_{i+1}$
  - If $H(\kappa_i) = s_i$, this means that there is no state-mismatch attack happening. We unlock our left lock with $k_i = \mathsf{HSoln}[\kappa_i]$.
  - Otherwise, there must be an attack happening. We construct a witness and create a key that embeds the witness verifiable with x_i. This gives us $k_i = \mathsf{WSoln}[\kappa_{i+1}, x_{i+1}, \{\Gamma_i\}]$, where $\Gamma_i = r_i||s_i||s_{i+1}||o_i$.
- Otherwise, U_{i+1} demonstrated that the sender attempted to defraud some U_j, where $j > i$ unlocked ℓ_{i+1} by presenting an inconsistency witness $k_{i+1} = \mathsf{WSoln}[\kappa_j, x_j, \{\Gamma_{i+1}, \Gamma_{i+2}, \ldots, \Gamma_j\}]$.
  - We can simply construct $k_i = \mathsf{WSoln}[\kappa_j, x_j, \{\Gamma_i, \Gamma_{i+1}, \ldots, \Gamma_j\}]$ where $\Gamma_i = r_i||s_i||s_{i+1}||o_i$. This transforms the witness verifiable with x_{i+1} to a witness verifiable with x_i.

Note that both cases are covered by Vf — it accepts and verifies both "normal" unlocks with HSoln-tagged tuples, and "inconsistency" unlocks with WSoln. Thus, even though Astrape is a composition of XorCake and HashOnion, the final construction fully "inlines" the two into the same flow of initialization, coin creation, and unlocking, with no separate procedure to process inconsistency witnesses. Unlocking continues backwards towards the sender until all the locks created in the previous step are unlocked. We have balance security — node U_i can unlock its left lock ℓ_i if and only if node U_{i+1} has unlocked ℓ_{i+1}, so no intermediaries can lose any money.

6.5 Security and privacy analysis

To prove security and privacy, we show that the checks done in the Init and Unlock phases fully prevent either violations of relationship anonymity through incorrect application of HashOnion or any violations of balance security, even when adversaries can arbitrary corrupt messages. Importantly,

we show that under the definition of relationship anonymity, an adversary cannot forge messages in such a way to "fake" the propagation of an inconsistency witness and break anonymity. We also see that just as in AMHL, committing to the entire path during initialization eliminates the wormhole attack.

6.5.1 Relationship anonymity

Out of our three desired properties, relationship anonymity is the most important. Unlike plain XorCake, Astrape achieves relationship anonymity in a less trivial fashion; in particular we must show that the HashOnion component cannot lead to deanonymization.

Thus, we present a proof that Astrape does achieve relationship anonymity. Note that within this proof, we assume that the sender U_0 and one particular U_i are honest according to the definition of relationship anonymity, while everybody else can be compromised by the adversary $\mathcal{A}$. We show that the only situation when information hurting relationship anonymity leaks is when U_0 is dishonest. Recall that relationship anonymity is only meaningful when U_0 is honest (otherwise U_0 can simply tell the adversary all the information), so this property *does not lose any anonymity* compared to systems like AMHL that unconditionally hide relationship-revealing information.

Recall that in $\mathcal{F}_{\text{anon}}$, messages are encrypted to the recipient, but the sender is hidden. Therefore, $\mathcal{A}$ cannot *read* the Init message sent from U_0 to U_i, but can *corrupt* (replace) it with a message of its own. This can prove problematic, because in the rest of our analysis of relationship anonymity we want to treat messages that U_i obtained through $\mathcal{F}_{\text{anon}}$ as authentic messages from U_0. However, if we prove that the adversary cannot gain any information from corrupting these messages, corrupting these messages cannot lead to a break in relationship anonymity, so we can then assume the messages are not corrupted. We also do not need to consider attacks where the adversary corrupts messages to the recipient or another intermediary, as the adversary is already allowed to directly corrupt those parties.

Lemma 6.5.1: The adversary cannot gain any information by corrupting $\mathcal{F}_{\text{anon}}$ messages sent by U_0 to the honest intermediary U_i.

Proof. Recall the Init message from U_0 to U_i is of the form $s_i^I = (r_i, s_i, s_{i+1}, x_i, x_{i+1}, o_i)$. Suppose the adversary replaces this message with $s_i^{I'} = (r'_i, s'_i, s'_{i+1}, x'_i, x'_{i+1}, o'_i)$. We first note that due to U_i's self-consistency check of $x'_i = H(r'_i||s'_i||s'_{i+1}||o'_i||x'_{i+1})$, any forgery of one variable requires the adversary either to *know* or *forge* every other variable. Therefore, after a corrupted Init, the only values U_0 wished to send to U_i that the adversary does not know — r_i, o_i — are destroyed. There is thus nothing in $s_i^{I'}$ now that U_i could reveal to the adversary in later stages of the protocol to possibly break relationship anonymity. □

Lemma 6.5.2: Given an honest sender and uncorrupted communication between it and the honest intermediary U_i, the left and right locks of U_i, ℓ_i and ℓ_{i+1}, are always unlocked normally and never by inconsistency witness. That is, the adversary cannot "frame" an honest sender to an honest intermediary.

Proof. There are only two cases where an honest U_i unlocks the left lock ℓ_i by inconsistency witness:

1. The right lock was unlocked normally, but with an XorCake solution κ where $H(\kappa) = s_{i+1}$ yet $H(r_i \oplus \kappa) \neq s_i$.
2. The right lock was unlocked with a witness recognized by HashOnion.

The first case is impossible with an honest sender, since it would imply finding a hash collision in H.

In the second case, under the random oracle model, where we can assume that H acts as a binding commitment scheme, this means that x_{i+1} commits to inconsistent information that violates the protocol. But since we know that U_0 is honest, such an x_{i+1} could not have been sent to U_i, so this is impossible as well.

Thus, with an honest sender, there cannot be any inconsistency witnesses used when unlocking ℓ_i and ℓ_{i+1}. □

Note: Lemma 6.5.2 *does not* claim that if U_0 is honest, no inconsistency witnesses can be generated at all — it only covers a very particular situation: the locks immediately adjacent to U_i in the case that communication is uncorrupted. In fact, two adjacent adversarial nodes U_j, U_k elsewhere can easily unlock the lock ℓ_k between them with a witness even when U_0 is honest by simply incorrectly executing Unlock (in case of off-chain payment channels) or replacing ℓ_k with a corrupt lock.

But can other inconsistency witnesses getting propagated break anonymity? Fortunately, no. First of all, dishonest intermediaries "acting out" an inconsistency witness with invalid locks between them cannot lead to a valid inconsistency witness that can unlock a *valid* lock, and thus cannot reveal any information that the dishonest intermediaries do not already know. The presence of these inconsistency witnesses therefore cannot lead to breaking anonymity.

A more interesting case is if the adversary *corrupts the locks around the honest intermediary*. After all, $\mathcal{F}_{\text{anon}}$ is not authenticated. Then, we would indeed be able to get an inconsistency

witness to propagate across the honest intermediary even when the sender is honest — this sounds dangerous!

The catch is that in corrupting the locks, the adversary must replace the internal state of the honest intermediary entirely with information the adversary already knows. This destroys any information the adversary might be interested in, and any further behavior in the network by definition cannot leak anonymity-harming information. We showed this in Lemma 6.5.1. Intuitively, it is *untampered* secret information from U_0 held by the honest intermediary U_i, which can only be revealed by an inconsistency witness "propagating across" U_i, that must be leaked to violate relationship anonymity.

We now proceed to formalize the notion that without this information, the adversary cannot break relationship anonymity.

Lemma 6.5.3: Given two simultaneous payments with the same value and paths $\cdots \to U_{i-1} \to U_i \to U_{i+1} \to \cdots$ and $\cdots \to U'_{i-1} \to U_i \to U'_{i+1} \to \cdots$, the attacker cannot distinguish this situation from two payments with paths $\cdots \to U_{i-1} \to U_i \to U'_{i+1} \to \cdots$ and $\cdots \to U'_{i-1} \to U_i \to U_{i+1} \to \cdots$.

Proof. We assume that the attacker $\mathcal{A}$ has corrupted all intermediate nodes before and after U_i. This means that in the Init phase, $\mathcal{A}$ learns all states s_j^I where $j \neq i$ for the first transaction, and also all states $s_j'^I$ where $j \neq i$ for the second transaction, but not the internal state of U_i, which is $s_i^{(\prime)I}$. The only values revealed to $\mathcal{A}$ that contain the internal state of U_i are commitments x_j where $j \leq i$ — but reconstructing U_i's initial state from x_j implies breaking the preimage resistance of H, which is impossible if H is a random oracle.

In the Create phase, we note that U_i obviously does not reveal any internal state except $s_{i+1}^{(\prime)}$ and $x_{i+1}^{(\prime)}$, which the attacker already knows from Init. Furthermore, by Lemma 6.5.2 we know that the two locks surrounding U_i can only be unlocked through the HTLC-based "normal" case, which again does not reveal more information about $s_i^{(\prime)}$.

Thus, the attacker's game reduces to guessing the predecessor and successor of U_i for both transactions, given all states $s_j^{(\prime)I}$ for $j \neq i$, with probability non-negligibly more than $1/2$ (as a function of λ). We can further reduce this to $\mathcal{A}$ guessing whether state s_{i-1}^I comes from the same transaction as s_{i+1}^I or $s_{i+1}'^I$; note that all of these states come from corrupt users adjacent to U_i.

$\mathcal{A}$ knows

$$s_{i-1}^I = (r_{i-1}, s_{i-1}, s_i, x_{i-1}, x_i, o_{i-1})$$
$$s_{i+1}^I = (r_{i+1}, s_{i+1}, s_{i+2}, x_{i+1}, x_{i+2}, o_{i+1})$$

$$s'^{I}_{i+1} = (r'_{i+1}, s'_{i+1}, s'_{i+2}, x'_{i+1}, x'_{i+2}, o'_{i+1})$$

$\mathcal{A}$ must then determine whether U_0 knows r_i, o_i, and κ_{i+1} such that ($x_i = H(r_i||s_i||s_{i+1}||o_i||x_{i+1})$ and $s_{i+1} = H(\kappa_{i+1})$) or else ($x_i = H(r_i||s_i||s'_{i+1}||o_i||x'_{i+1})$ and $s'_{i+1} = H(\kappa_{i+1})$), knowing that $s_i = H(r_i \oplus \kappa_{i+1})$. However, modeling H as a random oracle, these two cases are indistinguishable from $\mathcal{A}$, as required. □

Theorem 6.5.4: Astrape has relationship anonymity.

Proof. This follows immediately from the preceding lemmas. □

6.5.2 Balance security

We now show that Astrape has balance security. An interesting note is that balance security can be proven in a purely *local* way. As long as every intermediary can locally ensure that any right-lock solution can turn into a left-lock solution, no honest intermediary will end the protocol with less money than it started with, and balance security holds. Recall also from Section 6.4.5 that even the exceptional, inconsistency-witness-handling case fits into the same GMHL model — each intermediary U_i waiting for its right lock to unlock, and using the information revealed to construct a solution for its left lock that satisfies Vf. Thus, unlike the case with relationship anonymity, we do not need to consider the "rest of the world" other than U_i. Other participants and communication can be arbitrarily corrupt without affecting balance security.

Lemma 6.5.5: An honest user U_i always receives enough information to unlock its left lock ℓ_i if its right lock ℓ_{i+1} is unlocked.

Proof. We have two cases to consider.

In the first case, ℓ_{i+1} is unlocked by solving the XorCake component — with $k_{i+1} = \mathsf{HSoln}[\kappa_{i+1}]$.

Either κ satisfies $H(r_i \oplus \kappa) = s_i$ or it does not. If it does, U_i can construct $\kappa' = H(r_i \oplus \kappa)$ and use $k_i = \mathsf{HSoln}[\kappa]$ to unlock $\ell_i = \mathsf{Astrape}[s_i, x_i]$. If not, we have an inconsistency witness, and we can construct $k_i = \mathsf{WSoln}[\kappa, x_{i+1}, \{\Gamma_i\}]$ where $\Gamma_i = r_i||s_i||s_{i+1}||o_i$, and $H(\Gamma_i||x_{i+1}) = x_i$. By plugging into Vf, we see that this k_i is able to unlock ℓ_i.

In the second case, our right lock ℓ_{i+1} is unlocked by inconsistency using the HashOnion-like component. We can construct our own k_i to unlock ℓ_i by simply extending the witness with an additional Γ_i, which Vf will then accept. □

Theorem 6.5.6: Astrape has balance security.

Proof. For every intermediary U_i, its left lock has funds equal to its right lock (plus fees). Following Lemma 6.5.5, this means that any money lost by U_i's right lock being spent can always be recovered by spending its left lock. Thus, intermediaries cannot lose any money.

Therefore, Astrape has balance security according to its definition in Section 6.3.3. □

6.5.3 Wormhole immunity

The wormhole attack [Mal+19] is a generic attack on PCNs where given an honest sender and two corrupted nodes U_i, U_j, $j > i + 1$ separated by at least one honest node, U_i can unlock its left lock once U_j's right lock is unlocked. This "cuts out" all the participants in between, depriving them of fees from the sender.

Astrape is immune to the wormhole attack. This is because when U_{i+1} is honest, ℓ_{i+1} must be unlocked before U_i even has the information to unlock ℓ_i — information from "further down the line" at U_j where $j > i + 1$ is not sufficient. We note that Astrape fits the model given in the AMHL paper [Mal+19] for wormhole-immune PCN constructions, since the sender must know the complete path before sending money and there are two rounds of communication.

6.6 Practical concerns

6.6.1 Blockchain implementation

Astrape is easy to implement on blockchains with general scripting languages, like Themelio and Ethereum, as well as layer-2 PCNs such as Raiden built on these blockchains, but blockchains without Turing-complete scripts involve two main challenges.

First, these blockchains typically do not allow recursion or loops in lock scripts. This means that we cannot directly implement the Vf function. Instead, we must "unroll" Vf to explicitly check for witnesses to inconsistencies in the parameters given to U_i, U_{i+1}, etc. So for an n-hop payment the size of every lock script grows to $\Theta(n)$. In practice, the mean path length in the Lightning Network is currently around 5 (see our measurements in Section 6.7.5), and privacy-focused onion routing systems such as Tor or I2P typically use 3 to 5 hops. We believe linear-length script sizes are not a significant concern for Astrape deployment. An Astrape deployment can simply pick an arbitrary maximum for the number of hops supported and achieve reasonable worst-case performance.

The second issue is more serious: some blockchains have so little scripting that Astrape cannot be implemented. Astrape requires an "append-like" operation $||$ that can take in two bytestrings and

combine them in a collision-resistant manner. Unfortunately, the biggest blockchain Bitcoin has disabled all string-manipulation opcodes. Whether an implementation based solely on the 32-bit integer arithmetic that Bitcoin uses is possible is an interesting open question.

6.6.2 Side-channel deanonymization

We follow existing literature on anonymous PCNs [Mal+17; Mal+19] in pursuing relationship anonymity as our privacy goal. This means that a powerful adversary cannot distinguish between two payments sharing one honest intermediary that happen at exactly the same time with exactly the same values. However, real-life scenarios are rarely so clean, and violations of the assumptions of relationship anonymity can easily break privacy. For example, timing patterns may deanonymize users — perhaps only AliceCorp pays employees large lump sums on the 3rd and 14th of each month, letting the adversary detect AliceCorp employees.

Why, then, do we use relationship anonymity as our privacy goal? The main reason is that timing and value anonymity are largely *orthogonal* to relationship anonymity. Once we achieve relationship anonymity, we can adapt a large body of existing research in defeating side-channel attacks against anonymity systems like onion routing. Timing attack defenses can be directly lifted from works like Feigenbaum et al. [FJS10], while the information leaked by cryptocurrency values can be vastly diminished simply by using fixed denominations.

6.6.3 Griefing attacks

An unusual property of Astrape is that its worst-case cost (the HashOnion backup mechanism) diverges from its "average"-case cost (XorCake). Compared to HTLC, Astrape might open the door to *griefing* attacks, where an adversary sends coins to itself to intentionally burden victim intermediaries with large lock sizes and verification costs. These costs can be substantial,

as there are limits on the number of concurrent locks a single payment channel can support, imposed by the underlying blockchain.

Fortunately, both griefing based on verification costs and griefing based on lock sizes are easy to mitigate. In the first case, we can charge a penalty P for every hop the HashOnion case is used. For example, for each hop ℓ_i, in addition to the main payment coin encumbered by Astrape's composite XorCake/HashOnion lock, we can add a coin of value $P(n-i)$ which is unlockable by the same parameters that unlock the HashOnion case of the main coin. This way, if HashOnion is activated, every hop burdened by verifying HashOnion proofs can collect an extra P from its predecessor, which ultimately penalizes U_0.

Lock sizes are another possible way of griefing Astrape. For instance, the adversary can pay himself with a long path, but simply never unlock the coins and allow them to time out, burdening the victims with the cost of the locks. In Astrape, such a griefer does impose larger costs than in HTLC, as locks can be larger than in HTLC, but similar griefing problems occur in any system where failed transactions cannot incur fees, including HTLC.

But in case lock-size griefing becomes a problem, a robust defense that would also apply to other PCNs is to simply charge a small nonrefundable fee even in the case of timeout (for example, by refunding a small portion of the payment between U_i and U_{i+1} to U_{i+1} rather than U_i). This does allow an intermediary to pocket this small fee and not forward the payment, weakening balance security somewhat, but setting this fee smaller than the fee for actually forwarding the payment would likely make it unprofitable for intermediaries to do so.

Finally, an important point to note is that because Astrape messages are unauthenticated, *non-local* anti-griefing mitigations that attempt to punish griefers network-wide rather at the lock-script level are unlikely to work. In particular, it may be tempting to impose a network-wide penalty for senders who send inconsistent data that force intermediaries to generate expensive inconsistency witnesses, say through a reputation-based blacklist or destroying some staking bond. But in fact inconsistency witnesses cannot be tied to a specific sender who triggered them — there might not even be a "guilty" sender, since inconsistency might come from corrupted transmission (which as we argued in the security analysis cannot lead to deanonymization), and in any case the identity of the sender is hidden by $\mathcal{F}_{\text{anon}}$.

Fortunately, globally punishing nodes that send inconsistent messages is completely unnecessary for Astrape's security. As we discussed earlier, inconsistency witnesses are already fully handled by the normal Unlock and Vf GMHL functions, and do not require a separate recovery path or punishment mechanism to restore balance security. The additional burden that witnesses pose on network resources can also be fully mitigated through the hop-by-hop punishments that we just outlined.

6.7 Comparison with existing work

In this section, we compare Astrape with existing PCN constructions. First, we compare Astrape's design choices and features with that of other systems, showing that it explores a novel design space. Then, we evaluate Astrape's concrete performance. We compare Astrape's performance with that of other PCN constructions, both anonymous and non-anonymous. Finally, we explore Astrape's performance on a real-world network graph from the Lightning Network.

Table 6.1: Comparison of different PCNs

	Topology	Anon[a]	Efficient[b]	Crypto
HTLC	Mesh	No	Yes	Sig. + hash
Tumblebit	Hub	Yes	No	Custom RSA
Bolt	Hub	Yes	Yes	NIZKP
Teechain	Hub	Yes	Yes	Trusted comp.
Fulgor/Rayo	Mesh	Yes	No	ZKP
AMHL_{van}[c]	Mesh	Yes	Yes	Homom. OWF
AMHL_{ecd}	Mesh	Yes	Yes	ECDSA, Homom. enc.
AMHL_{sch}	Mesh	Yes	Yes	Schnorr sigs
Blitz	Mesh	Weak[d]	Yes	SA[e]sig. + hash
Astrape	Mesh	Yes	Yes	Sig. + hash

[a] Relationship anonymity

[b] Roughly comparable performance to HTLC. For example, ZKPs requiring many orders of magnitude more computation time than HTLC are not considered "efficient".

[c] AMHL is a family of three closely related constructions. We denote by van, ecd, sch the "vanilla", ECDSA, and Schnorr implementations respectively.

[d] See discussion in Section 6.2.4.

[e] A "stealth-address" signature scheme; i.e., a signature scheme where any party knowing a public key can generate unlinkably different public keys that correspond to the same private key.

6.7.1 Design comparison

In Table 6.1, we compare Astrape's properties with those of existing payment channel networks. We see that except for HTLC, which does not achieve anonymity, all previous PCN networks use cryptographic constructions specialized for their use case. Furthermore, only more recent constructions achieve efficiency comparable to HTLC. It is thus clear that Astrape is the first and only PCN construction that works on all PCN topologies, achieves strong anonymity, and performs at high efficiency, while using the same simple cryptography as HTLC.

Table 6.2: Resource usage of different PCN systems (n hops, c-byte HTLC contract, d-byte Astrape contract); AHML variants van,ecd,sch as in Table 6.1

	Plain HTLC	Fulgor/ Rayo	AMHL	Astrape (Bitcoin Cash)	Astrape
Comput. time (ms)	< 0.001	$\approx 200n$	$\approx n$ (van) $\approx 3n$ (ecd) $\approx 3n$ (sch)	$\approx 0.7n$	$\approx 0.25n$
Comm. size (bytes)	$32n$	$1650000n$	$32 + 96n$ (van) $416 + 128n$ (ecd) $256 + 128n$ (sch)	$192n$	$192n$
Lock (bytes)	$32 + c$	$32 + c$	$32 + c$	$108 + 39 \cdot n$	$64 + d$
Unlock, normal case (bytes)	32	32	32 (van) / 64	32	32
Unlock, worst case (bytes)	32	32	32 (van) / 64	$64 + 128 \cdot n$	$64 + 128 \cdot n$

6.7.2 Implementation and benchmark setup

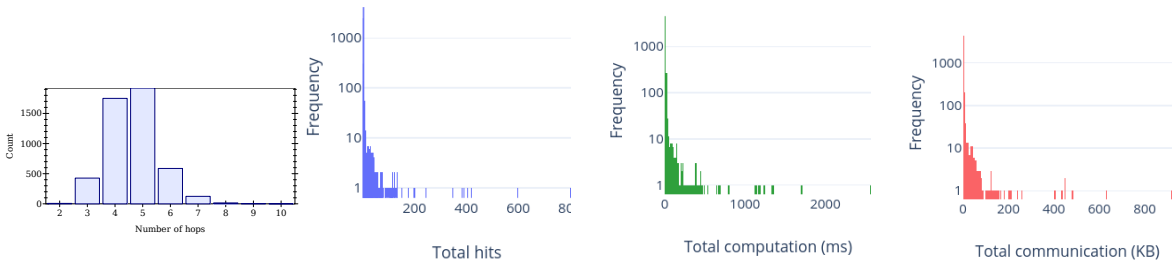
To demonstrate the feasibility and performance of our construction, we developed a prototype implementation in the Go programming language. We implemented all the cryptographic constructions of Astrape inside a simulated GMHL model. We used the libsodium library's implementation of the ed25519 [JL17] signature scheme and blake2b [Aum+13] hash function. In addition, we generated script locks in Bitcoin Cash's scripting language to illustrate script sizes for scripting languages with no loops.

All tests were done on a machine with a 3.2 GHz Intel Core i7 and 16 GB RAM. Network latency is not simulated, as this is highly application-dependent. These conditions are designed to be maximally similar to those under which Fulgor [Mal+17] and AMHL [Mal+19] were evaluated, allowing us to compare the results directly.

6.7.3 Resource usage

Our first set of tests compares Astrape's resource usage to that of other PCN constructions. We compare both a simulation of Astrape and a concrete implementation using Bitcoin Cash's scripting language to traditional HTLC, Fulgor/Rayo, and all three variants of AMHL.

We summarize the results in Table 6.2, where n refers to the number of hops, c to the size of an HTLC contract, and d to the size of an Astrape contract. We copy results for Fulgor/Rayo [Mal+17] and AMHL (ECDSA) [Mal+19] from their original sources, which use an essentially identical setup.



(a) Shortest paths (b) Hit distribution (c) Computation (d) Communication

Figure 6.2: Overhead distribution

Computation time. We measure computation time, with communication and other overhead ignored. The time measured is the sum of the CPU time taken by each hop, for all steps of the algorithm. We note that by eschewing non-standard cryptographic primitives, Astrape achieves lower computation times across the board compared to Fulgor/Rayo and AMHL.

Communication overhead. We also measure the communication overhead of each system. This is defined as all the data that needs to be communicated *other than the locks and their opening solutions*. For example, in Astrape, this includes all the setup information sent from U_0, while in AMHL this includes everything exchanged during the Setup, Lock, and Rel [Mal+19] phases. We see that Astrape has by far the least communication overhead of all the anonymous PCN constructions. Note especially the extreme overhead of the zero-knowledge proofs used in Fulgor/Rayo.

Lock overhead. The last measure is *per-coin* lock overhead — the size of each lock script (the "lock size") and that of the information required to unlock it (the "unlock size"). This is a very important component of a system's resource usage, since lock and unlock sizes directly translate into transaction fees in blockchain cryptocurrencies. Astrape's performance differs in two important ways.

First of all, Astrape's Vf function is expressed in a manner that requires nested-hashing a dynamic-length series of values. In blockchains like Bitcoin Cash that support neither recursion nor loops in their scripting language, we must "unroll" the Vf implementation. This causes lock sizes to be linear in the number of hops. In blockchains with general-purpose scripting languages, though, lock size is generally constant. Second, the worst-case unlock size is larger for Astrape. When the sender is malicious and all coins have to be spent by invoking HashOnion, we need n parameters $(\Gamma_1, \ldots, \Gamma_n)$ to unlock each coin for an n-hop payment. However, despite this asymptotic disadvantage, we

believe that Astrape nevertheless offers competitive lock performance. This is because payment routes are quite short in practice, as we will shortly see.

6.7.4 Privacy vs. overhead

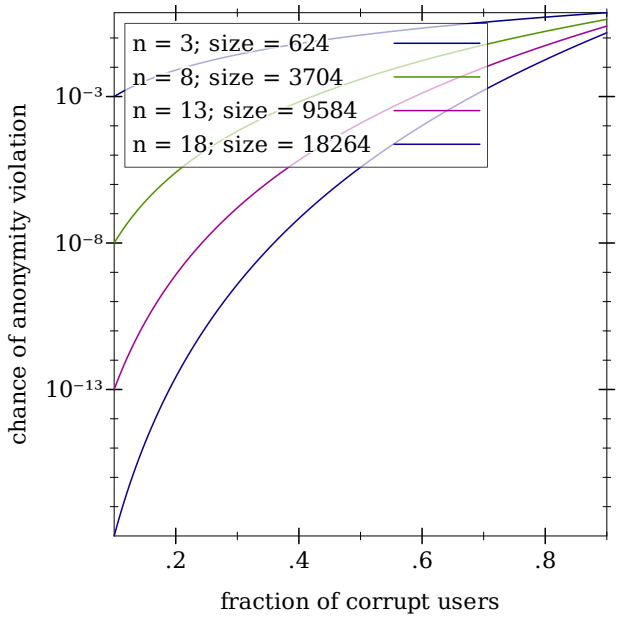


Figure 6.3: Privacy at different values for n and Bitcoin Cash lock script sizes

One possible concern with Astrape's performance behavior is that it establishes a tradeoff between privacy and performance. As n increases, we get higher privacy but also higher per-hop overhead, especially for Bitcoin Cash-like blockchains with no support for recursive lock scripts. We next quantify this effect to see whether or not overhead poses a serious barrier to high levels of privacy.

To do so, we calculate the relationship between the fraction of corrupt users and the probability that at least one intermediary in n hops would be honest, for varying values of n. This is plotted in Figure 6.3. We see that by increasing n to 8 we can reduce the chance of compromise to less than 1% even if most of the nodes are corrupt, while still having script sizes less than 10 KB. Although such a script will be hundreds of times the size of an HTLC script, it is still small enough that communicating it across payment channels will not be a large burden.

These script sizes do, however, cause large transaction costs in case transactions must settle on the blockchain. Thus, Astrape is probably not very suitable for on-chain coin-mixing services

for blockchains with no recursive or looped lock scripts. Other anonymous multi-hop transaction constructions, like the Fulgor or AMHL, may be more suitable in those situations.

6.7.5 Statistical simulation

Finally, we simulate the performance of Astrape on the network graph of the Lightning Network (LN).

Setup. We set up a mainnet Lightning Network node using the lnd [19] reference implementation. We then used the lncli describegraph command to capture the network topology of Lightning Network in Feburary 2021. This gives us a graph of 9566 nodes and 72164 edges. Finally, we randomly sample 5,000 pairs of nodes in the network and calculate the shortest paths between them. This gives us a randomly sampled set of real-life payment paths.

Path statistics. As we have previously shown, paths more than 10 hops long still have fairly small overhead even with non-recursive lock scripts, but much longer paths will cause rather large unlock sizes. We examine whether the graph topology will force payments to grow too long; Figure 6.2a illustrates the distribution of lengths for our 5,000 randomly selected payment paths. On average, a payment path was 5.12 hops long, though the Lightning Network specification allows up to 20 hops. This indicates that shortest payment paths long enough to pose seriously ballooning worst-case overhead are practically nonexistent.

Total scalability. One important attribute we wish to explore with the LN topology is the total scalability of the network — how fast can a PCN process transactions as a whole.

To do so, we keep track of how many times each node appears, or is "hit", in our 5,000 randomly selected payment routes. On average, this is 2.99, but the vast majority of nodes are hit only once, while a few nodes are hit hundreds of times. The distribution of hits is plotted in Figure 6.2b as a log-linear histogram. We then look at the *distribution of overhead* in the network for both computation and communication. This is by calculating the total computation and communication cost for each node "hit" by the 1,000 random payments, using values from the Bitcoin Cash implementation.

Computation cost is plotted in Figure 6.2c. We see that the most heavily loaded node in the entire network did around 2,000 ms of computation

to process 1,500 transactions. This indicates that the largest hubs in a PCN with the current Lightning Network topology will be able to process around 750

transactions a second per CPU core. Such a throughput is orders of magnitude higher than that of typical blockchains and is within reach of many traditional payment systems. We note that this is only the maximum throughput of a *single CPU core* — in practice hubs likely have multicore machines, and with many hubs the total LN throughput will be many times this number.

Communication cost is plotted in Figure 6.2d. We pessimistically assume that all payments are settled through HashOnion. Even so, the total network load averages to only about 3.58 KB per node. The largest hubs' total load still does not exceed 1 MB. This illustrates that the bottleneck is actually computation, not communication.

In summary, we see that Astrape's worst-case asymptotic performance poses no barriers to the total throughput of an Astrape-powered payment channel network. PCNs can enjoy the superb scalability associated with them just as easily with Astrape-powered privacy and security.

6.8 Future work

6.8.1 Constant-space lock sizes

One interesting problem is whether or not an anonymous multi-hop transaction protocol using only "conventional" cryptography can work with constant lock size. Avoiding worst-case $O(n)$ lock sizes can allow the use of anonymous PCNs in applications where these lock sizes are problematic, such as on-chain coin mixing or bandwidth-constrained environments. However, it seems unlikely that constant space usage is possible with Astrape-type constructions based on multi-hop HLTC and inconsistency witnesses. There does not seem to be a straightforward way of securely committing to rightward (r_i, s_i) parameters with sublinear witnesses.

We wish to investigate as future work whether or not more sophisticated cryptographic commitment mechanisms, such as Merkle trees, can be used to eliminate the linearly growing inconsistency witnesses.

6.8.2 More robust notions of privacy

Privacy properties not captured by relationship anonymity, such as immunity from timing and other side-channel attacks, are notoriously hard to model.

Unfortunately, anonymity breaks in anonymous communication systems have largely been attacks on these side channels. For instance, the Silk Road 2.0 darknet market was deanonymized largely from timing attacks [Mur15]. In fact, for payment channel networks these side channels

probably matter even more. Payment volume is a small fraction of anonymous communication volume, making "innocent" transactions that an adversary can confuse targeted ones for rare.

We urgently need a notion of payment channel privacy that captures the "messy" concepts of time information and distinguishability from other payments across the whole network — not just distinguishability between two payments of the same length, time, etc., intersecting at the same honest intermediary. More research into this area could lead to a much more robustly anonymous payment channel network that can truly replace less scalable alternatives such as physical cash or on-chain payments with zero-knowledge-proof based cryptocurrencies like Zcash.

6.9 Conclusion

First-generation payment channel networks and other trust-minimizing intermediarized cryptocurrency payment systems lack strong privacy and security guarantees. Existing research, although solving the privacy and security problems, tend to rely on custom cryptographic primitives that cannot be easily swapped with alternatives based on different computational hardness assumptions.

We presented Astrape, a novel PCN construction that breaks this conundrum. Astrape is the first PCN that achieves relationship anonymity and balance security with only black-box access to generic conventional cryptography. It relies on a general idea of using non-anonymous post-hoc inconsistency witnesses to achieve balance security, while avoiding any information leaks when senders are not corrupt. This allows Astrape to avoid dealing with the zero-knowledge verification used to achieve balance security in existing relationship-anonymous PCNs without sacrificing any anonymity or security properties.

Furthermore, we demonstrate that Astrape is practical to deploy in the real world. Performance is superior on average to existing private PCNs, even on blockchains that are unsuitable for free-form smart contracts. We also showed that Astrape achieves high scalability on a real-world payment channel network graph.

Chapter 7

Bitforest: efficient blockchain PKI

In this chapter, we introduce another mid-level protocol important to a wide range of applications: Bitforest, a public key infrastructure (PKI) protocol. Public key infrastructures (PKIs), or more generally secure naming systems, lie at the foundation of the security of any communication system. Without a trustworthy binding between user-facing names, such as domain names, and cryptographic identities, such as public keys, all security guarantees against active attackers come crashing down like a house of cards.

We originally formulated Bitforest in 2017 [DKB18] as a protocol built on Bitcoin; its main innovation was the trick of encoding an on-chain key-value store as a binary search tree embedded in Bitcoin's UTXO graph — this gives us the first known way of mapping names to public keys on Bitcoin that can be efficiently queried even by Bitcoin's rudimentary light clients, which is an interesting demonstration of the inherent flexiblity and power of the UTXO/coin-graph model.

Since Bitcoin's features are almost a clean subset of Themelio's, for most of this chapter we will be presenting Bitforest in its 2017-era context as a Bitcoin-based PKI. At the end, we will describe how Themelio's much more powerful light clients and scripting allow for further simplifications and improvements in Bitforest.

7.1 Introduction

Nowadays, cryptographic protocols such as TLS, which provide secure communications over insecure networks such as the Internet, have gained extremely pervasive deployment. It would not be an understatement to say that without these secure channels, use of the Internet could not have expanded into fields such as private communication and e-commerce. Yet the security of these

protocols relies ultimately on one thing: *a secure public key infrastructure*. Application-level names must be somehow bound to cryptographic identities, such as public keys, in a trustworthy way; otherwise, security against active man-in-the-middle attacks cannot be achieved.

Unfortunately, building a secure PKI has proved to be quite difficult. In fact, naming systems were once conjectured to be subject to a tradeoff known as *Zooko's triangle* [Kam11], namely, that the three properties of *human-meaningful*, *decentralized*, and *secure* cannot be achieved simultaneously. Most traditional systems, like the hierarchical PKI used in TLS and S/MIME, attempt to achieve security and human-meaningfulness by introducing trusted third parties, such as certificate authorities (CAs) or key servers. Though compromised or incompetent trusted parties have repeatedly damaged security [Del+14; NB13], Zooko's triangle seemed to imply that there could be no better alternative.

The advent of blockchains, public append-only ledgers that are unforgeable yet fully decentralized, heralded the demise of Zooko's triangle. Though the first blockchain, Bitcoin [Nak08], was initially conceived only as a cryptocurrency for financial purposes, its first-ever fork, Namecoin, pioneered the idea of building a secure naming system by encoding name-value pairs inside a blockchain. The blockchain is used as an append-only global log of state transitions to provide consensus on the mapping of names to cryptographic identities without trusting any third party; any changes to name-value pairs are broadcast in new blocks and appended to the globally replicated log. Several newer blockchain naming system designs, such as Certcoin [FVY14], follow the same general design.

However, though Namecoin-like designs bring significant security improvements, they also face many new challenges. All nodes in the network must synchronize and validate a local copy of the blockchain, so anybody wishing to look up names in a secure fashion must face large, linearly-increasing storage costs. Additionally, without a large user-base, blockchains are vulnerable to attacks which compromise their security guarantees; in fact, Namecoin is known to be subject to a 51% attack — a devastating reduction in blockchain security — due to its small number of miners [Ali+16]. Finally, deploying any new features to a Namecoin- like system is very difficult, as users must all agree to run a newer version of the protocol within a short period of time to maintain the distributed consensus.

Newer blockchain-based PKIs, such as Blockstack [Ali+16] and ENS [Ser23], do attempt to mitigate these issues. However, although Blockstack makes it easier to deploy new features and reduces the amount of data that needs to be replicated to all participants by moving most of the data away from the underlying blockchain, it still fails to eliminate the requirement for verifying large amounts of blockchain data, and continues to be much less flexible in enforcing rules for namespaces compared to centralized solutions.

These issues with existing blockchain-based distributed PKIs motivate us to build a new system, Bitforest, which aims to sidestep the common pitfalls of purely distributed blockchain networks by using a hybrid architecture with a minimally-trusted centralized service encoding information

crucial to the integrity of the PKI in a novel data structure embedded in an arbitrary cryptocurrency blockchain. This allows us to inherit the security of a blockchain while retaining much of the properties of traditional PKIs including fast name lookups, low storage requirements, policy flexibility, and performance. We believe that by escaping the apparent hard tradeoff between decentralized, blockchain-anchored trust and high performance, Bitforest makes it significantly more practical to deploy a blockchain-based PKI.

This paper makes the following contributions:

- We present a novel insert-only, efficiently queryable binary search tree consisting of cryptocurrency transactions that can be ported to almost any cryptocurrency blockchain. This blockchain-neutral *index tree* allows key-value bindings to be inserted and values to be appended to, but not other modifications; these constraints are enforced directly by the underlying blockchain's double-spending prevention, and cannot be violated without forking the blockchain.
- We outline the design of Bitforest, a next-generation blockchain PKI based on combining an index tree embedded in the Bitcoin blockchain with a centralized *name administrator* managing a namespace, acting as a "gatekeeper" for all modifications and queries to the namespace. This allows enforcement of policies in areas such as access control in a fine-grained manner unavailable to completely decentralized systems while retaining security for registered names equivalent to that of blockchain-based PKIs.
- We show, through both real-world experimental results and numerical simulation, that Bitforest is highly efficient, with performance comparable to centralized systems, both in storage overhead and lookup cost. It is the first blockchain-based PKI to offer to low-overhead light clients security as strong and decentralized as that of expensive full nodes in previous systems.

7.2 Background and related work

In this section, we discuss the background to the development of Bitforest — in particular, our definition of a secure naming system and previous work on using blockchains to improve upon traditional centralized-trust PKIs.

7.2.1 Secure naming systems

It is important to explicitly define the criteria by which we call the guarantees provided by a naming system "secure". Unfortunately, the security goals of different PKIs are often vaguely described,

causing confusion on exactly what sort of guarantees a system attempts to give. Thus, instead of referring simply to "security", we discuss two separate concepts in this paper: *policy enforcement* and *identity retention*.

Policy enforcement refers to the enforcement of *externally-defined* constraints on bindings in the namespace. For example, in the traditional PKI used in TLS, CAs checking for domain ownership before issuing domain- validated certificates is an example of policy enforcement: "domain ownership" is a concept defined by DNS, which is not itself covered by TLS's PKI. Another common example is real-world identity certification, as in TLS's extended validation certificates, where having a name in a namespace certifies claims about real-world facts like business registration.

Identity retention, on the other hand, is a purely *internal*, self-consistency property. As defined by the authors of Certcoin [FVY14], it is simply the inability for anybody to impersonate an identity already registered to somebody else. That is, any bindings in the namespace can only be changed by authorization by the current owner, and it should not be possible to fool anybody into accepting forged bindings that disagree with the original binding registered to a name. Any recovery mechanism (e.g. some backup key for changing the name in case credentials are lost) would in this case also require the explicit prior consent of the name holder (e.g. by signing a backup key with a primary key).

Violations of identity retention tend to be far more devastating to secure communications than violations of policy enforcement. Man-in-the-middle attacks inherently require an adversary to make unauthorized changes to the public key bindings of a name, thus breaking identity retention. On the other hand, failure of policy enforcement is rarely a disaster, unless a system relies solely on policy enforcement to provide its security guarantees, with no concept of "already registered" names, such as in TLS's PKI.

7.2.2 Need for decentralized trust

Traditionally, enforcing the two security properties mentioned above was the responsibility of trusted, centralized third-party entities, such as CAs or key servers. For example, in the CA-based PKI used in TLS, CAs must check every request for a certificate to execute the policy enforcement that CAs are required to do — if a CA makes a mistake, either unintentionally or maliciously, then the PKI's security guarantees break down.

Unfortunately, such compromises of trusted third parties and subsequent collapse of PKI security are not rare: most famously, in 2011 the DigiNotar certificate authority was compromised [PC11], leading to large amounts of malicious, yet widely accepted, certificates being issued. Similar incidents have occurred in centralized systems from key servers to software update signing systems [PP17], and even honest central authorities often prove to be very lax in actually enforcing their claimed

security policies [PC11], continuing to undermine trust in the security of centralized PKI. Clearly, eliminating or reducing central points of trust in a PKI would greatly help it achieve stronger security.

One category of naming system that promises to eliminate all central points of trust uses blockchains — append-only ledgers maintained by a secure distributed consensus algorithm. We take a look at the origins of blockchain- based PKI, then evaluate state-of-the-art blockchain naming systems which attempt to achieve better security and performance goals, while examining their strengths and lingering weaknesses.

7.2.3 First-generation blockchain PKIs

Unlike PKIs that use centralized parties such as key servers or certificate authorities, naming systems based on embedding name information in a blockchain are fundamentally decentralized. All participants in a blockchain form a peer-to-peer network that maintains an append-only, growing log of transactions; without changing the consensus of a majority of the network, past blocks cannot be rewritten. Importantly, blockchains typically use economic incentives to encourage selfish participants, or "miners", to honestly contribute large amounts of resources to maintaining and verifying the append-only log, making it very difficult for attackers to amass enough resources to hijack the network consensus. PKIs based on blockchains hold promise in achieving distributed trust for identity retention, though without any centralized governance, policy enforcement is often unsupported.

Namecoin is the first blockchain-based naming system, and it is implemented as a separate cryptocurrency based on the Bitcoin code [Nak08], designed to function as an alternative to the Domain Name System (DNS). As the first fork of the Bitcoin software, Namecoin adds to the blockchain a name-value store that maps domain names to DNS records [Kal+15]. Identity retention of names is protected, as there is a single canonical log of events in the blockchain, and all name updates are signed by the name owner. Rewriting past entries in Namecoin is difficult and requires vast amounts of computational resources due to the use of the Hashcash proof-of-work, which is also used by Bitcoin [Nak08].

Namecoin and similar systems, such as Certcoin [FVY14], that embed naming information in a dedicated blockchain, though, pay a very heavy price in performance, scalability, and flexibility due to the requirement for all participants to replicate and maintain a consensus on the global append-only log. More specifically, Namecoin, and systems that embed name mappings directly in dedicated blockchains in general, have the following problems:

Slow writes New transactions in the Namecoin blockchain must wait for several minutes to a few hours before they are accepted in a block, due to the high latency inherent to distributed consensus based on proof-of- work[11]. Thus, new names registered in Namecoin would only become queryable after a few minutes or hours.

Limited throughput The block size limits the total number of transactions that can be recorded in a block. If a pending name registration does not get written in the next block, it has to wait for another block to be created. For Bitcoin and Namecoin, the current block size is 1 MB [Nak08] or approximately 1,000 transactions.

Lack of policy enforcement As noted earlier, Namecoin's security model abandons policy enforcement entirely, choosing only to enforce identity retention. In practice, this has resulted in large amounts of name-squatting and other forms of abuse — in fact, the overwhelming majority of Namecoin names contain no useful data [Kal+15].

Bootstrapping cost To securely query a domain name, one needs to participate in the blockchain network and setup a local Namecoin server [Nam], requiring synchronization of the entire blockchain. This bootstrapping process transfers a large volume of data — 5.13 GB as of October 2017. Though this may not be a big problem for a server wishing to run a naming service for multiple clients over a protocol like DNS — and be trusted blindly by those clients — for users who want to securely query domain names for applications such as web surfing, storage of the unwieldy blockchain prevents Namecoin from being an appropriate solution.

Scale-dependent security Namecoin's, like any proof- of-work blockchain, relies on the existence of a large, diverse population of miners to deliver its promised security and lack of centralized trust. Unfortunately, this population cannot exist without widespread adoption of Namecoin; in practice, Namecoin is not widely used, resulting in most of the mining power centralized in a single small mining pool, a devastating failure in a proof-of-work blockchain's security. [Ali+16]

7.3 Newer blockchain PKIs

In light of the abovementioned issues present in Namecoin, quite a few newer naming systems have appeared that still anchor trust on the robust decentralized conensus of blockchains. These blockchain PKIs share in common a goal to improve security by piggybacking on existing, more

secure blockchains while increasing performance, but attempt to solve Namecoin's problems in a plethora of different ways. We examine some newer blockchain-based systems particularly influential on Bitforest's design.

Blockstack is arguably the state of the art in fully distributed blockchain-backed PKI. It is a fairly complex system that attempts to provide a general-purpose naming and storage system, but its main innovation is the usage of a blockchain-neutral "virtualchain", where cryptocurrency transactions embedded within an existing blockchain, such as Bitcoin, define the state of the namespace in place of a separate blockchain. Anybody can add transactions to the virtualchain simply by broadcasting appropriately-formatted transactions on the underlying public blockchain. This mechanism for announcing state changes in the namespace allows Blockstack nodes to come to a consensus on the total order of namespace changes, and therefore the status of the namespace, simply by trusting an existing blockchain [Ali+16].

The layer of indirection given by virtualchain makes Blockstack blockchain-neutral. Blockstack can anchor its security on any blockchain — since it writes name operations to the blockchain as on-chain transactions, tampering with name-value pairs would require rewriting the underlying blockchain. For this reason, though it previously used Namecoin, the Blockstack naming system now uses Bitcoin [Ali+16], currently the biggest and most reliable blockchain, as the underlying blockchain to maximize security.

Blockchain neutrality also allows Blockstack to implement features not found on the underlying blockchain. For example, Blockstack partially solved the problems regarding bootstrapping cost by enabling lower storage costs and more secure light clients with consensus hashes, a concept not found in Bitcoin. Nodes calculate a consensus hash — a cryptographic hash of a concatenation between a Merkle hash of virtualchain operations and a geometric series of previous consensus hashes [Ali+16]. This allows Blockstack nodes to quickly check whether they have the same view of the global state and validate blocks that have occurred previously, while allowing light clients with access to a trusted source for the latest consensus hash to quickly validate the authenticity of a previous name operation with a later consensus hash, a process known as Simple Name Verification (SNV) [Ali+16].

Unfortunately, Blockstack still fails to adequately address the high bootstrapping cost of blockchains: thin clients need a trusted consensus hash in order to lookup names securely, which can only be obtained by a trusted full node, forcing security-critical applications to run full nodes, which must scan the entire 100+ GB blockchain. Furthermore, like Namecoin, policy enforcement is unavailable due to the entirely permissionless design of the namespace.

Another class of newer blockchain-backed PKI, while similarly encoding namespace change as blockchain transactions, centralizes administration over the namespace, allowing flexible policy enforcement and the easy utilization of blockchain-specific properties to reduce bootstrapping costs

while retaining distributed trust for identity retention. These PKIs combine centralized control for aspects such as access control and indexing with properties enforced by blockchains beyond simple transaction ordering to define a secure yet efficiently queriable name mapping. Unlike Blockstack, they allow light clients to still achieve high performance while retaining strong decentralized trust, eliminating the cumbersome bootstrapping of Namecoin and Blockstack — arguably the most significant performance barrier to adoption of blockchain PKIs.

EthIKS is a good example of this class of blockchain PKI. It is based on CONIKS, a transparency-based PKI combining a centralized key server with semi-decentralized "auditors" (checking that the server shows the same bindings to everyone) and "monitors" (checking that updates do not violate identity retention) to reliably detect malicious behavior by the key server, rather than proactively prevent them from happening. EthIKS inherits the benefits of CONIKS' centralization in policy enforcement, but instead of relying on monitoring and auditing to detect malicious behavior after the fact, the Ethereum blockchain is used to guarantee identity retention and self-consistency of the system. This is accomplished through implementing CONIKS as an Ethereum smart contract that stores hashes of namespace changes and checks the validity of each operation. As long as the Ethereum blockchain enforces its own validation rules, the smart contract of EthIKS guarantees that it cannot ever record information that would violate identity retention. Highly secure light clients are supported, since Ethereum allows lightweight clients to quickly validate the state of smart contracts [Woo14] while downloading only a small fraction of the blockchain (the blockchain headers). This brings EthIKS very good security properties, with both identity retention backed by the smart contract, and policy enforcement done by the centralized provider. However, EthIKS, though not requiring its own blockchain, is tightly coupled to the Ethereum blockchain, using its Turing-complete smart contracts and in-built key-value storage system [Bon16]. Its reliance on these very powerful but highly idiosyncratic features of the Ethereum blockchain makes the concept practically impossible to generalize to other blockchains, tying its performance and security to a specific blockchain.

Another example is a system proposed by the authors of Catena [TD17], an efficiently queriable and centrally-managed append-only log embedded in the Bitcoin blockchain, where a Catena log is combined with CONIKS to ensure providers cannot equivocate, eliminating the need for third-party monitors, significantly improving the distributed trust of CONIKS. This "CONIKS+Catena" PKI still relies on self-monitoring and reactive, transparency-based security, falling short of the security guarantees promised by blockchain-based PKIs. However, Catena does have a significant advantage: although the authors do not discuss it in detail, Catena is much more blockchain-neutral than EthIKS: instead of relying on smart contracts as its blockchain-specific, efficiency-enhancing validation system, it simply exploits double-spending protection, essentially the constraint that the same money cannot be spent twice, a property strictly enforced by almost every cryptocurrency. This allows "CONIKS+Catena" to be easily ported to a variety of blockchains while keeping its

security and performance properties.

7.3.1 Towards a better system

Blockchain-based naming systems make large strides towards stronger security compared to traditional, centralized-trust PKIs, yet important unsolved problems remain. In particular, blockchain-based systems have severe limitations in policy enforcement, performance, scalability, and flexibility. Although newer blockchain systems exist that attempt to address these issues with conventional blockchain PKIs, they still are quite far from our ideal of a decentralized secure naming system, either inheriting much of the performance issues of traditional blockchain PKIs (Blockstack), requiring a blockchain with generalized smart contract support (EthIKS), or failing to fully anchor security in the blockchain's decentralized trust (Catena+CONIKS).

Evidently, we desire not only the flexibility and performance of centrally-managed naming systems like traditional PKIs but also the strong security found in blockchain networks based upon trust distributed among many self-serving participants. Moreover, in light of the security implications and poor flexibility caused by tightly binding our system to a particular blockchain, we wish to have a blockchain-neutral design that can easily be ported to any compatible blockchain that best provides the performance and security required. No existing system achieves this combination of performance competitive with traditional systems, strong security, and the choice between wide variety of blockchains to provide robust decentralized trust.

It is clear that we need a new system, which although building on existing ideas such as semi-trusted centralized administration, blockchain-independent architecture, and utilizing blockchain rules to validate operations, has a completely different overall design. With a novel architecture unlike the typical design of blockchain PKIs, combining a centralized namespace administrator with an efficiently and securely queryable dictionary data structure embedded in an arbitrary cryptocurrency blockchain, Bitforest aims to achieve exactly that.

7.4 Design

We have designed Bitforest to implement a secure naming system supporting both policy enforcement as well as identity retention, as defined previously, with the more important property of identity retention fully backed by the security guarantees of a cryptocurrency blockchain. In this section, we describe Bitforest's design principles, its overall architecture, and how its goals are achieved by its design.

7.4.1 Design principles

Based on the experiences of previous blockchain-based PKIs, we designed Bitforest in accordance with the following principles:

Blockchain neutrality Although our prototype implementation uses Bitcoin as its underlying blockchain, Bitforest is designed to not rely on any particular blockchain. In fact, any public blockchain with its security based on the principle of spending transaction outputs at most once (the "no double spending" rule) can be easily plugged in. This allows individual deployments of Bitforest to adapt to whatever blockchain fits the application the best — for example, Bitcoin can be used for applications needing high security with less concern over update throughput, while private blockchains such as Hyperledger can be used for scenarios where fully decentralizing trust is less important than high performance.

Centralized administration We want it to be possible to create centrally-managed namespaces, such as directories of employees belonging to a certain organization. Administrators are responsible for such policy enforcement, failure of which by itself cannot lead to the most devastating attacks. This is in contrast to fully distributed blockchain-based PKIs such as Namecoin and Blockstack, which lack policy enforcement altogether and thus are often vulnerable to name-squatting and other abuse — the vast majority of name-value bindings in Namecoin, for example, are in fact spam containing no useful information [Kal+15]. Furthermore, as the experiences of both EthIKS and Catena — both of which have a centralized component — show, a centralized provider can often be used to increase performance by indexing the blockchain and maintaining in-blockchain data structures.

Decentralized identity retention On the other hand, we need to avoid trusting the central administrator when enforcing a bottom-line of security — identity retention. That is, even with a malicious administrator, it should not be possible to make changes to name-value bindings without authorization from the owner of the name, or for anyone to be fooled into obtaining an incorrect binding. Robustly decentralized identity retention is in fact perhaps the raison d'être of blockchain-based PKIs, as other systems almost always require at least partially trusting some centralized entity.

7.5 Architectural overview

Bitforest uses a client-server architecture, where namespace administrators, or NAs, administer namespaces; clients then look up names in a certain namespace by querying information published by a particular NA. Instead of imposing a global infrastructure of NAs, Bitforest allows developers to set up application-specific NA infrastructures, similar to how traditional PKIs may have application-specific root CAs.

In Bitforest, names in a certain namespace are mapped by NAs to unique indices in a deterministic, verifiable way. Clients then use an index tree, a novel insert-only dictionary data structure embedded inside the blockchain, to map these indices to an append-only list of operation hashes — secure hashes of each element of the operation log, an NA-provided history of all values ever bound to the name. Each entry in the operation log also contains signatures by a cryptographic identity declared in the previous entry; in effect, updates to the name describe who is authorized to append further updates. The last entry in the operation log defines the current binding of the name.

Catena's innovation of exploiting the blockchain-neutral constraint of double-spending prevention is borrowed in Bitforest to secure the insert-only property of Bitforest's index tree, applying the same concept to a much more powerful data structure than a simple linked list. This allows Bitforest to achieve strong identity retention backed by a fundamental security guarantee of its underlying blockchain while allowing efficient and secure lookup of names without prohibitive bootstrapping cost. Furthermore, storing only indices and hashes in the in-blockchain data structure allows policy enforcement to be controlled entirely by the NA. Updates and queries must involve the NA, as only the NA can compute indices from names or provide the actual operations whose hashes are stored in the index tree.

7.5.1 Threat model

Trusted bootstrapping: We assume that Bitforest clients are able to securely obtain the root transaction of the index tree, which acts as a unique identifier of a Bitforest NA. It is crucial to know that, with Bitforest's model of individual NA-defined namespaces, it is only meaningful to talk about the enforcement of security properties given a particular namespace defined by its index tree root. The exact mechanism by which clients are informed of the NA's root transaction is orthogonal; since it is a fixed and static value, one easy method is to simply hard-code a deployment-specific value in the client. We stress that any naming system must start with fixed, trusted bootstrapping data — blockchain PKIs, for example, must be able to connect to the correct blockchain, generally by hard-coding its first block.

Adversarial name administrator In our threat model, the name administrator can be malicious or compromised, and it wishes to break identity retention. That is, the adversarial NA wishes to fool clients into associating an existing name to an adversary-chosen value without the consent of the owner of the cryptographic identity authorized in the operation log to update the name. In addition, we assume that an adversarial NA may have no need to hide its attacks to succeed in its goals, so attacks do not need to be hidden forever.

Arbitrary policy enforcement We do trust, however, that the name administrator executes policy enforcement correctly, for some application-specific definition of correctness. This is seemingly an overly strong assumption, but in practice, it is impossible to distinguish an honest NA which capriciously changes its policies with no warning from an NA compromised to circumvent its intended policy. In other words, we simply define any arbitrary behavior in allowing or forbidding updates or lookups as an NA's policy, while a situation that prevents the NA from executing these actions is instead treated as a compromise of the system.

No double spending We assume that the adversary is unable to attack the underlying cryptocurrency blockchain of a Bitforest PKI in such a way that two transactions which send the same money to different destinations both appear to be valid. It is generally accepted that the distributed consensus algorithms of popular public blockchains such as Bitcoin and Ethereum are highly robust to any adversary with a reasonable amount of resources, and double spending by the adversary is thus vanishingly unlikely to succeed, especially in the long run.[1]

Secure verification of transaction existence We assume that there is a way for lightweight clients not participating in blockchain replication to confirm that a given transaction exists in the blockchain. Furthermore, we assume that the adversary is unable to fool a client to falsely believe that a transaction exists on a blockchain.

7.6 Basic structure of the index tree

Bitforest's index tree is essentially an insert-only binary search tree (BST) consisting of transactions in the underlying blockchain, mapping indices, 256-bit keys corresponding uniquely to names (see Figure 7.1), to 256-bit cryptographic hashes of every entry in an operation log. Every node in the index tree is a transaction, where:

[1] Short-range forks are normal and expected in many blockchains, but blockchains typically guarantee that on-chain data eventually becomes economically infeasible to revert.

- The first input, for all nodes except the root node, spends either the 1st or 2nd output of the node's parent in the index tree.
- The first four outputs are, in order,
  - A spendable output to be spent by the left child
  - A spendable output to be spent by the right child
  - A spendable output to be spent by the update chain
  - An unspendable output (for example, using OP_RETURN in Bitcoin) storing an index and the 256-bit hash of the first operation of the name associated with the index

(The "update chain" that uses the third output of each node stores subsequent updates to the name, and is discussed in Section 7.6.1) All the transactions in the index tree represent nodes in a BST, sorted by index, with smaller indices to the left and larger indices to the right; children are related to their parents by spending a particular output in their parent. Money flows down the tree from the root transaction and is used to cover transaction costs; additional funds can be introduced at any point through transaction inputs other than the first one, while unnecessary funds can be reclaimed by using outputs other than the first four. Fig. 1 gives an illustration of a small index tree. Inserting a new index-hash pair to the tree is done in a straightforward manner — a new node is broadcast to the blockchain, spending the output that would connect it to the tree in such a way that searching for the index would find the new node. The NA maintains control over inserts to the index tree by sending the spendable outputs to addresses whose private keys only the NA knows.

An important property is that every node can have at most one left node, and at most one right node, confirmed in the blockchain. This is due to double-spending prevention — each transaction output can be spent by one and only one subsequent transaction — and ensures that existing nodes and links in the index tree cannot be overwritten once created as long as the blockchain's guarantees hold. Thus, the BST represented by the index tree can only be added to, and indices, once bound to a particular hash, can never be rebound to anything else.

More crucially, for any index in the index tree, we can generate a short proof of existence that the index is bound to a particular 256-bit hash. This proof consists of the transactions that form the path, traced by applying the standard BST search algorithm for the index[2], leading from the root transaction to a transaction that includes the given index. Such a proof does not take up much space: for a tree with randomly-distributed indices, the length of a proof of existence is expected to be

[2]On Bitcoin, this requires full knowledge of the blockchain content, which we assume the party *building* the proof does have. As we will see, *verifying* the proof does not require searching through the whole blockchain, or indeed accessing any blockchain data other than the root hashes.

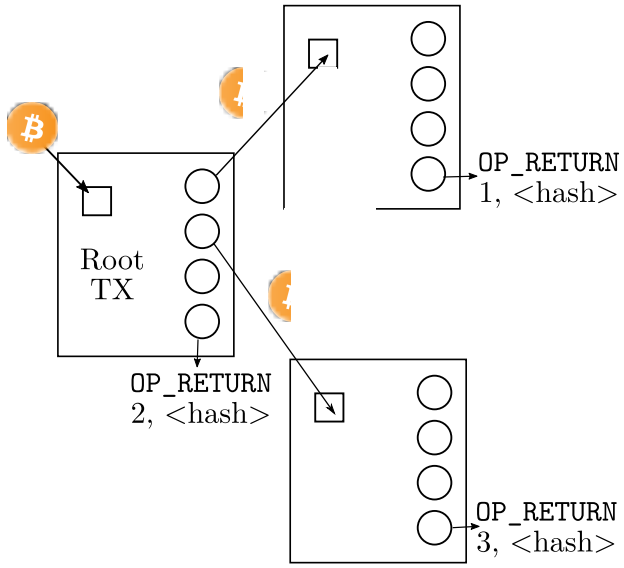


Figure 7.1: Example of an index tree with indices 1,2,3

$\Theta(\log n)$. As an example, in the index tree in Figure 7.1, the root hash and its left child constitute a proof that the index 1 is bound to a certain hash.

Any Bitforest client, with hardcoded knowledge of the root transaction, can validate a proof of existence in an index tree for an index x, by checking the following properties:

1. The transactions indeed exist in the blockchain.
2. The first transaction in the proof is indeed the already-known root transaction.
3. The fourth output of the last transaction binds a hash to x.
4. Each transaction in the proof t_i spends the expected output of the previous transaction t_{i-1}. That is, if x is smaller than the index in the fourth output of t_{i-1}, t_i spends its first output while if x is bigger it spends the second output; in case x is equal to the index of a transaction not at the end of the path, the proof is declared invalid.

The last property is, essentially, to check that the proof of existence really ends at the "canonical" transaction for a certain index — the transaction on which a step-by-step search from the root would terminate. Malicious NAs may attempt to insert non-canonical transactions into the tree which refer to the same index, violating the BST invariant, but paths from the root transaction to

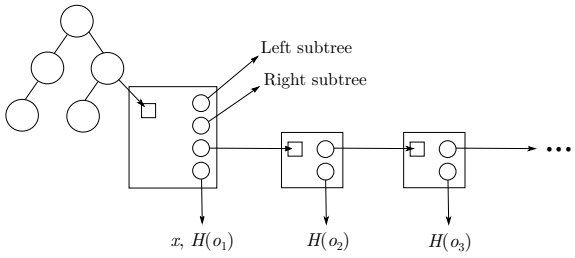


Figure 7.2: Example of an update chain associated with index x

these transactions would not satisfy the last property, preventing the NA from being able to show a proof of existence for them, safeguarding the insert-only property of the index tree.

In addition, checking these properties takes very little time and storage space. Checking that the transactions are confirmed in the blockchain can be done efficiently and securely by a lightweight light client in essentially any cryptocurrency blockchain. For example, for Bitcoin, this only requires the client to synchronize the Bitcoin block headers — a very small amount of data compared to the entire blockchain — using Simple Payment Verification (SPV) [Nak08]. The remaining three steps are done locally by the client and complete essentially instantly.

7.6.1 Updating an index tree

In the previous section, we sketched the general structure of the "tree" part of the index tree, which provides an insert-only, efficiently queryable mapping from numerical indices to 256-byte hashes embedded in the blockchain. Yet in Bitforest, names are not mapped to a fixed immutable piece of data, but rather to an operation log that can always be appended to. How is the gap between the two abstractions bridged?

The answer is the update chain, a Catena transaction chain [TD17] spending the 3rd output of a tree node transaction. A Catena chain is the first, and possibly the simplest grow-only blockchain data structure based on exploiting double-spend prevention, and provides an append-only linked list of log entries. In Bitforest's case, the update chain is used to store hashes of operation log entries other than the first one; the first operation's hash is stored inline in the tree node to reduce the number of transaction needed to register a new name.

Figure 7.2 illustrates an index x bound to an operation log $o_1, o_2, \ldots$. By appending to the growing update chain, the NA can append more entries to the operation log; the underlying blockchain's double spend prevention forbids the NA from either rolling back the log to an earlier state or overwriting existing log entries, providing the abstraction we want: a mapping of keys to append-only

operation logs.

The proofs of existence discussed in the previous sub- section can easily be extended to the entire operation log: we simply include all the transactions forming the update chain in the proof. Clients then verify that each transaction in the update chain does indeed spend the previous entry's first output, and that the first transaction in the update chain spends the third output of the tree node recording the correct index. The size of the proof is now $\Theta(\ell + \log n)$, where ℓ is the length of the update chain, and n is the number of names in the namespace. In practice, ℓ is unlikely to be large; names bindings in PKIs are typically changed only due to infrequent events such as name transfer or key revocation that may not even happen for the majority of names.

7.6.2 Mapping names to indices

We have discussed how the index tree securely maps numerical indices to cryptographic hashes of operation logs. But how are these "indices" related to the names in the namespace? After all, the whole point of a Zooko's-triangle-violating PKI is to provide human-readable names; requiring clients to look up random numeric indices defeats the purpose.

One naive solution is to map names to indices with a cryptographically secure hash function, such as SHA-256. However, this direct approach is unworkable, as it allows anybody with access to the blockchain to check the existence of names in the namespace without involving the NA, facilitating name enumeration and severely weakening policy enforcement over the area of access control over listing names. In addition, malicious name registrants would be able to greatly degrade the performance of the service by registering names with indices that when inserted in order would grossly unbalance the index tree, causing the size of some proofs to be exorbitantly large.

The solution to both of these problems is to compute indices using a verifiable random function (VRF) [MRV99], which is a random function that requires a private key to compute, but can then be publicly verified. One example of a VRF is VXEdDSA, which Bitforest uses in practice. Given such a function VRF(), the index *x* for a name *n* is computed as:

$$x = \mathrm{HMAC}(\mathrm{VRF}_{K_{\mathrm{VRF}}}(n), n)$$

where $\mathrm{HMAC}()$ is an HMAC using SHA-256, and K_{VRF} is a public key belonging to the NA already known to clients of that NA. Using a VRF directly as an index would open up the possibility of a malicious NA registering two names n, n_0 mapping to the same index — most VRFs used in practice do not guarantee collision-resistance given knowledge of the private key — so the result is wrapped in an HMAC with the name as the key to take advantage of the collision resistance of a conventional cryptographic hash.

Using a VRF solves both of the problems mentioned above. Firstly, name lookups now must go through the NA, as only the NA can compute the index using a VRF and walk the index tree to generate a proof of existence for that index. This eliminates the policy enforcement and name enumeration issues — both CONIKS [20] and EthIKS [16] use a similar construction to prevent name enumeration. Secondly, we obtain randomly distributed indices that are verifiable after the fact, but unpredictable by anybody other than the NA before a name is registered and placed in the index tree. Thus, malicious registrants cannot precompute the indices of names and register them in a pathological order that would unbalance the tree. Malicious NAs, of course, can arbitrarily unbalance the tree, but we already assume trust in the NA for availability, even though we distrust it for identity retention.

Now we finally have all the details to describe a full response by the NA to a client's query for a name n:

- A VRF output $\mathrm{VRF}_{K_{\mathrm{VRF}}}(n)$, which the client verifies and from which it derives the index x
- A proof of existence for the operation log bound to x, which the client checks according to the procedure given in Section 7.6, containing a cryptographic hash of every element in the operation log
- The operation log itself, which the client checks does indeed hash to the values given in the proof of existence

7.7 Operation logs and identity retention

The index tree, combined with a VRF-based function mapping, now gives us a way of securely obtaining an append-only history of any name, which can only be appended to through the NA — the operation log, consisting of many individual operations. Each operation in the operation log must be signed by a cryptographic identity declared in the previous operation, preventing any changes to a name's binding unauthorized by the owner of the name. This subsection will discuss the details of how the operation log works.

7.7.1 Structure of an operation

Each operation contains the following fields:

- A random nonce

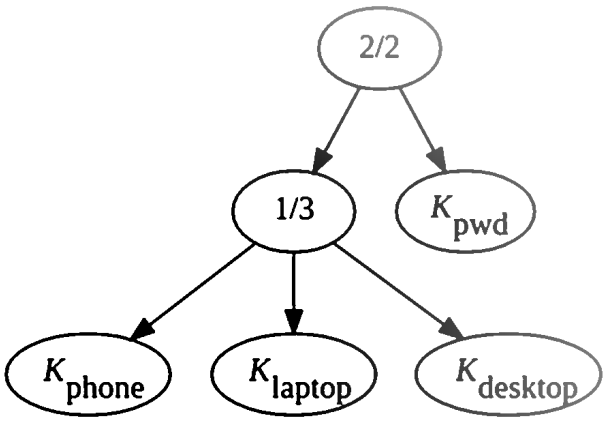


Figure 7.3: Example of an identity script

- An identity script representing the cryptographic identity authorized to generate the next operation bound to the name, i.e. the "current owner"
- A collection of cryptographic signatures, valid with respect to the identity script declared in the previous operation
- Data associated with name

The random nonce is used to prevent replay attacks, where the NA replays previous bindings in cases where a previous binding has signatures from the latest owner. It also randomizes the hash of each operation, preventing the operation log hashes in the index tree from leaking any information about the namespace to anybody without access to the NA.

7.7.2 Identity scripts and signatures

Identity scripts are the entities representing cryptographic identities in Bitforest. They encode a tree structure of key quorums; every identity script is either:

- An Ed25519 public key, or
- n out of m identities

Figure 7.3 depicts a hypothetical identity of a user in a chat application, where the PKI manages important public profile details, such as name and application-level public key. Our example user has

enabled two-factor authentication for his or her account, so changing important data must require authorization using both the password and a device; the Bitforest identity itself elegantly encodes this requirement.

We use a simple stack-based scripting language, inspired by payment scripts in Bitcoin [Nak08], to represent these trees; the identity in Figure 7.3, for example, is written as `<K_phone> <K_laptop> <K_desktop> .quorum 1. 3. .ed25519 <K_pwd> .quorum 2. 2.`. The ith operation o_i in an operation log is only valid if the signatures on o_i are valid with respect to the owner identity script declared in o_{i-1} .

We note here that nothing prevents this identity script system from expressing an entirely NA-trusting update policy that gives up identity retention for convenience, relying instead on transparency to implicitly regulate the NA's actions, similar to the behavior of "normal users" in CONIKS [Mel+15]. This can be done simply by assigning a keypair controlled by the NA, or even the identity script `.quorum 0. 0.`, as the "owner", which would allow the NA to append whatever log entry it wants without cryptographic signatures from the user of the name.

7.7.3 Summary

In this section, we presented an architectural overview of Bitforest. A novel data structure, the index tree, allows for the implementation of a securely and efficiently queryable index-value mapping embedded inside a generic cryptocurrency blockchain, where indices can only be inserted and not deleted, and values can only be appended to and not overwritten. This mapping then serves as the basis for a naming system that provides a unique combination of the strong, distributed-trust identity retention guarantees of existing blockchain-based solutions, and the highly flexible policy enforcement found in traditional centralized PKIs.

7.8 Attacks and mitigations

We have claimed that Bitforest gives us strong security within our threat model: policy enforcement is trustworthy given a trustworthy NA and identity retention is guaranteed given that the underlying blockchain is secure. Unfortunately, these assumptions might not always hold given certain powerful adversaries. We examine how Bitforest behaves under these attack scenarios, and how applications using Bitforest can mitigate the resulting damage.

7.8.1 Stale data attack

Malicious NAs may attempt to present out-of-date information to clients looking up names, by serving clients outdated proofs of inclusion in the index tree. For example, for a name with multiple updates in the operation log, the NA may cut off the last few entries of the update chain, fooling the client into accepting a binding that has been already invalidated by subsequent updates to the operation log. The NA can even give different stale bindings to different clients, making them disagree on the latest binding to a name.

This is not an attack against identity retention as defined earlier, but rather against non-equivocation — the property that everybody sees the same values bound to the same names — so it does not prevent Bitforest from achieving its security goals. The NA cannot show any bindings that have not been authorized by the name owner even if such an attack is carried out, and in practice such an attack is unlikely to cause significant security problems.

It is possible for Bitforest clients to at least partially defend against this attack. In our prototype implementation of Bitforest on the Bitcoin blockchain, we attempt to ascertain whether proofs of inclusion are up to date by querying well-known third parties whether the transaction output of the last entry of the proof that would extend the proof — for example, the first transaction output of the last element of the update chain — is indeed unspent. This forces the adversary to compromise these third parties in addition to the NA, significantly raising the difficulty of a successful attack.

Finally, this attack is non-viable if a blockchain allows secure verification by light clients of the current status of a sum of money, rather than simply the existence of transactions — Themelio is an example of such a blockchain. Such a feature would allow Bitforest clients to check whether transaction outputs are spent or not with the full assurance of the blockchain consensus; we chose to not include it in our blockchain model as most blockchains do not provide this functionality.

7.8.2 Blockchain attacks

Although we assume that the underlying blockchain of Bitforest is impervious to any attack that jeopardizes double-spending prevention, in practice no blockchain is completely secure. For example, proof-of-work blockchains can be subverted by attackers with more than half of the mining power of the networks (known as a 51% attack) [Nak08], and permissionless blockchains in general are often subject to eclipse attacks [Hei+15], where a highly adversarial network cuts a client off from the blockchain consensus and feeds it a view of a completely attacker-controlled fork. A broken blockchain that allows double-spending would allow an adversarial NA to build and show to clients multiple index trees claiming arbitrarily different bindings for names, subverting identity retention.

We do note, though, that even in the worst case of a blockchain allowing arbitrary double spending, such an attack would be very difficult to conceal — any evidence of two conflicting index trees would be immediate and undeniable proof that the NA is compromised. In addition, if at any point the blockchain's double-spending protection is restored, the adversary-controlled NA would no longer be able to mount attacks. Finally, a subverted blockchain cannot do anything to damage the security of a Bitforest namespace if it does not collude with a compromised NA.

7.9 Implementation and evaluation

In this section, we describe our implementation of Bitforest and evaluate Bitforest against existing naming systems, both by measuring their performance using experiments and by numerical simulation. We also discuss the cost of operating a Bitforest namespace across popular public blockchains.

7.9.1 Implementation

We created a prototype reference implementation of Bitforest in the Java programming language. The Bitforest library allows applications to easily create their own NAs and look up names securely; by default it uses the Bitcoin blockchain to store the index tree.

One particular area of implementation deserving some discussion is the ease of porting Bitforest to different blockchains. Although originally we implemented Bitforest for Bitcoin, in order to evaluate how well Bitforest achieves our goal of blockchain neutrality, we ported it to Litecoin, a well-known cryptocurrency "altcoin" with similar semantics to Bitcoin. Implementing the Litecoin version of Bitforest proved to be very easy — based upon spending transaction outputs like the majority of cryptocurrencies, Litecoin allows us to encode the index tree exactly as we have described it.

7.9.2 Lookup reference

In our first quantitative experiment, we evaluate the performance of doing lookups in Bitforest. As a comparison, we also evaluate the performance of secure client-server lookups in Blockstack, the current state-of-the-art in blockchain-neutral blockchain-based PKIs. Unlike most other blockchain-based naming systems, Blockstack has a secure thin-client lookup system — SNV — with at least some decentralized trust, allowing a contest between two systems with comparable security.

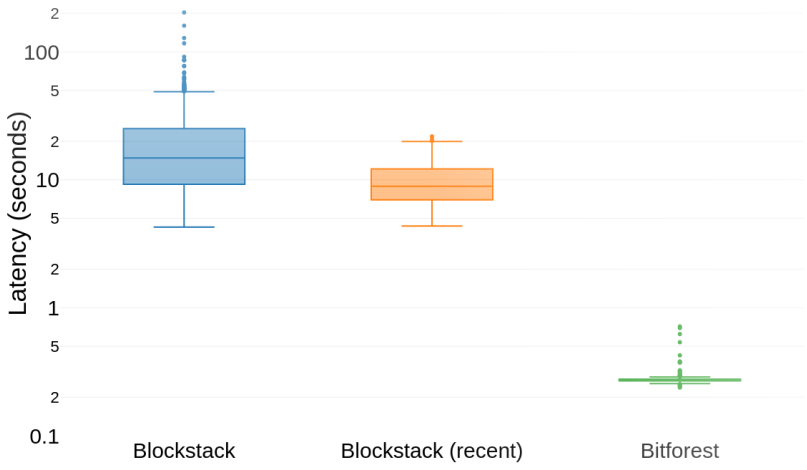


Figure 7.4: Lookup latency, compared with Blockstack

Both a Blockstack full node and a Bitforest NA are installed on a server, and a client with around 90 ms of network latency to the server is used to benchmark the two systems. We create a Bitforest NA — using the Bitcoin "testnet" [Nak08] to avoid exorbitant transaction costs — and insert 73,000 names, the approximate amount of names in Blockstack. Then, we query our Bitforest NA with 300 random dummy names previously placed in the namespace, while for Blockstack we use SNV to verify 300 random existing name records in the operational Blockstack network; in both cases, we sample the namespace without replacement. Total latency and bytes transferred are then measured by tracing network packets using Wireshark, avoiding inaccurate measurements due to application startup latency.

Figures 7.4 and 7.5 summarize the results of this experiment. Note that Blockstack lookups grow significantly slower as we query names registered in older and older blockchain blocks, as the SNV process requires the client to iteratively "walk" across old blocks until it hits the one where the name is registered; thus, we have a separate metric for Blockstack names registered after the start of 2017, and until mid-2017, called "Blockstack (recent)". [3]

We see that in both metrics, especially latency, Bitforest performs much better than Blockstack. The particularly lopsided difference in latency measurements likely stems from Blockstack's SNV implementation, which requires the client to incrementally walk backwards along Blockstack's

[3]Of course, 2017 is no longer recent. But 2017 data is still relevant, as Blockstack in its original form became unmaintained and recent data would not be very meaningful.

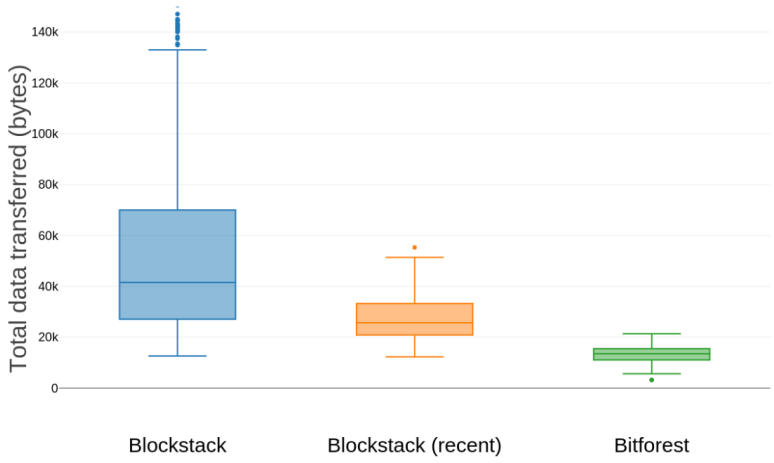


Figure 7.5: Data transferred per lookup, compared with Blockstack

virtualchain, taking up a large amount of network round trips to complete the procedure; on the other hand, Bitforest NAs give full proofs of existence for their index trees in a single request-response cycle. In particular, doing SNV proofs for old Blockstack names can take up to minutes.

Even considering this implementation flaw, though, Bitforest still proves significantly more efficient — this is illustrated by the comparison of data transferred, a metric less affected by issues with the implementation. The advantages of Bitforest's very simple and efficient index tree, as opposed to Blockstack's virtualchain, clearly show, allowing Bitforest queries to transfer only around 10 KB of data, while Blockstack SNV lookups use up several times or even an order of magnitude more bandwidth. Quite evidently, Bitforest's lookup procedure is far more performant than Blockstack's state-of-the-art blockchain-based secure thin-client lookup.

Furthermore, Bitforest offers acceptable speed even compared to systems with centralized trust. Figure 7.6 compares lookup latency between Bitforest and three systems lacking distributed trust: unsecured DNS, DNSSEC-secured DNS with all signature validation done by the client, and Blockstack's default server-trusting mode. For DNS, we look up a list of US government DNSSEC-enabled domains using the dig tool and its +sigchase option, while for Bitforest and Blockstack we use the same set of random names from the first experiment. We see that though DNS is very fast due to its aggressive caching, compared with client-verified DNSSEC and server-trusting Blockstack, both of which are less amenable to caching by the ISP, Bitforest offers excellent performance, even though it has fully-verifying clients and distributed trust.

In conclusion, we see that the price Bitforest pays in lookup performance to achieve much

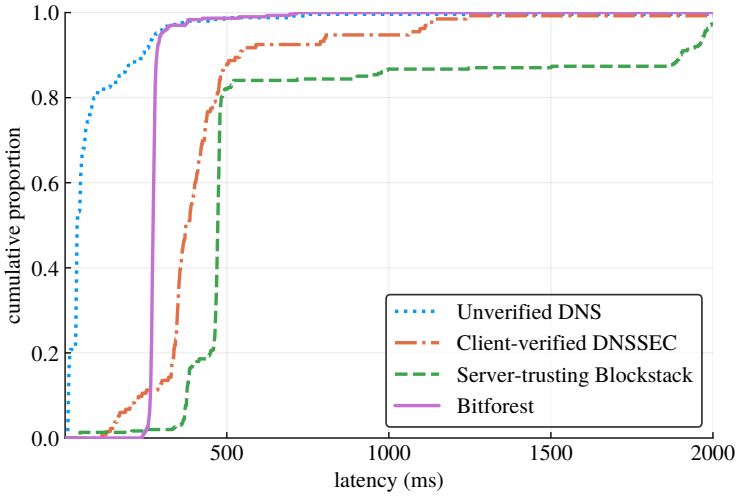


Figure 7.6: Bitforest lookup latency, compared with centralized-trust systems

stronger security is quite small, and should not be problematic for the vast majority of applications.

7.10 Conclusion and Themelio implementation

In this chapter, we presented Bitforest, a naming system with blockchain-backed security. Bitforest uses a new architecture informed by the successes and shortcomings of existing blockchain-based systems such as Blockstack and EthIKS, combining a centralized lookup service trusted only for policy enforcement with a novel, blockchain-neutral data structure built from cryptocurrency transactions that provides an efficiently queryable mapping from names to cryptographic hashes of their respective bindings. By doing so, Bitforest ensures that the central administrator cannot do anything that violates identity retention without breaking the security guarantees of the underlying cryptocurrency blockchain while allowing high flexibility in areas such as access control and privacy. Experimental results show that Bitforest performs markedly better than existing blockchain-based systems, with faster lookup than light clients and dramatically reduced storage overhead compared to full nodes; we also demonstrated that its performance penalty compared to traditional centralized PKIs is small to nonexistent.

An interesting final note is that unlike on Bitcoin, Bitforest can be implemented as a completely permissionless naming system without any "name administrator" on Themelio. This is by using a Melodeon covenant to enforce the shape of the index tree, an index tree structure that does not require a VRF (for example, a fixed-depth trie rather than a BST), and a decentralized off-chain

storage network like a DHT to store naming data. This would remove the policy enforcement opportunities that a name administrator provides, but on the other hand, provide much stronger censorship resistance.

Chapter 8

Sketches of applications

Astrape and Bitforest are two detailed examples of mid-level endogenous trust protocols that can be built on top of Themelio. In this chapter, we will take a look at some high-level sketches of complete, user-facing applications built in an "idiomatically Themelio" manner. We start from conventional "blockchain apps" like token games, working our way to complex "full-stack" apps like social networks. All of these gain high endogenous trust by interfacing with Themelio, yet many of them do so very indirectly through mid-layer protocols and do not resemble "blockchain applications" in the usual sense.

8.1 Token systems

Perhaps the most obvious application of Themelio is tokens, like fundraising tokens, NFTs, and new cryptocurrencies. Right now, a new token is most commonly implemented as a single stateful smart contract on Ethereum or a similar blockchain storing account abalances, following API standards like ERC-20 that define operations such as transferring tokens from one account to another. This way of implementing a token, unfortunately, has several major problems. First of all, managing the contract state is rather prone to error, often leading to critical security vulnerabilities (such as arithmetic errors while making transfers!). "Legibility" to light clients is also a significant concern, since even though the interfaces for other smart contracts calling the token contract on-chain follow standard APIs, these interfaces cannot be easily called by light clients, which must directly query the blockchain state and guess where the contract actually places information such as account balances.

In Themelio, on the other hand, custom tokens are extremely easy to create and transact, since they are a first-class concept in its coin-based data model. Creating a new Themelio-based token

relies on a special case in Themelio's transaction verification logic. Themelio must be balanced by currency — the total values of mels, syms, etc. in the input coins spent must be equal to the total values in the outputs — with the exception that an unlimited number of coins, with unconstrained values, can be created with *unlabeled* units. Outputs with unlabeled units will then create coins in the blockchain state with a new unit derived from the unique ID of the transaction.

Thus, a new cryptocurrency token can be created simply by creating any regular transaction while tacking on an additional unlabeled output with the value set to the maximum supply of the new token. This coin's constraint script will then determine the rules of token issuance — no other transactions can create coins with the same unit, since custom tokens are always denominated by the first transaction that created them.

As an example, imagine Foobar wants to create a new token, FooCoin. FooCoin would be sold to the public at a fixed rate of 1000 micromel per coin. Foobar would broadcast a transaction, say with ID `0xdeadbeef`, with an unlabeled output with an inexhaustibly large value (say 2^{64}) and the following Melodeon covenant:

```
// first output sends the rest of the FooCoins
//   and repeats this constraint
// second output sends FooCoin to buyer
// third output sends mels to Foobar
let spender = env_spender_tx() in
let foocoins_bought = env_parent_value() - spender.outputs[0].value in
spender.outputs[0].covhash == env_self_hash() &&
spender.outputs[0].denom == custom_denom(env_self_hash()) &&
spender.outputs[1].value == foocoins_bought &&
spender.outputs[1].denom == denom_foocoin() &&
spender.outputs[2].value == foocoins_bought * 1000 &&
spender.outputs[2].denom == denom_mel() &&
spender.outputs[2].covhash = foobar_address()
```

Anybody can then spend this FooCoin-denominated output, diverting some of the FooCoin to himself, leaving the rest with the same constraint, and giving FooBar mels in compensation. FooCoins not encumbered by this constraint freely transact using the same rules as syms and mels do, with no complex smart contracts needed to handle all the cases. Wallet software, payment processors, etc. can simply check for coins denominated in "`0xdeadbeef`" to support FooCoins.

Analogous constraints can be used to implement more complex rules for fungible cryptocurrency tokens. Non-fungible tokens, on the other hand, are implemented simply by creating a new token unit with a coin that has a value of 1. Since 1 cannot be further subdivided, this means that only one coin of that token can ever exist.

We see that unlike the rest of the possible applications in this overview, tokens are most naturally implemented without intervening abstractions. This is simply because Themelio is designed in a coin-centric way, and coins are the basis for building any token system.

8.2 Micropayment-incentivized peer-to-peer overlays

Peer-to-peer *overlay networks*, consisting of nodes and connections between them through the Internet, are an important building block for decentralized systems like IPFS, BitTorrent, Freenet, and similar systems.

A Themelio-based protocol ecosystem allows for P2P overlay networks with much better performance and security by solving several important issues:

Anti-sybil mechanisms Preventing fake or spam nodes from joining is a big problem for peer-to-peer networks. But using Themelio, we can solve it in a variety of ways, many of which are found in projects like Nym [DHK21]: on-chain proof-of-stake (requiring nodes to lock up funds to join a network), antispam governance (allowing holders of a certain token to vote for or against the membership of nodes), etc. [1] The main contribution of Themelio's model is its light clients, which allow P2P nodes to verifiably query on-chain state — often indirectly through implementations of protocols like Bitforest — without either running their own blockchain nodes or trusting gateways like Infura.

Incentivizing participation A promising avenue for incentivizing peer-to-peer networks is through *tit-for-tat micropayments*, where any action by one node that burdens another's resources (e.g. by transmitting data) must come with a mutually agreed payment. Payment channel networks like Astrape are an ideal medium for the extremely low-value payments; Themelio's design is important for allowing participation in these payment channel networks without centralized trust assumptions or heavyweight blockchain full nodes.

End-to-end encryption with friendly names A Themelio-based naming system, similar to the decentralized version of Bitforest described earlier, can allow participants of these overlay networks

[1] Of course, without a decentralized identity network, "plutocratic" token-based systems cannot solve problems that require *distinct people* to be in control of nodes, such as ensuring that different hops on an onion routing network belong to different people. But they are an effective way of implementing DoS-resistance, and to some extent, shareholder-like governance, especially if these tokens accrue value from the correct operation of the protocol.

to bind their public keys to human-readable names. This would allow contacting nodes on the overlay network to have a smooth user experience similar to using DNS on the traditional Internet. For instance, instead of entering long random-looking strings into a browser to access the equivalent of a Tor [DMS04] onion service, we can simply enter something like `example.mel`.

8.3 Private blockchains with endogenous trust

For some applications, nothing short of a new blockchain with its own consensus would do. Traditionally, this would require deploying a new blockchain, private or public, just for use within the application. But this greatly reduces security, as compromising a blockchain formed by consensus between a small number of people is much easier than taking over a public blockchain like Themelio or Ethereum.

A common solution in previous blockchains is to use a *metachain*, also known as a *virtualchain*, where every time a transaction happens on the smaller blockchain, it is embedded into a corresponding transaction on a public blockchain. Clients of the metachain then scan the entire public blockchain for valid-looking transactions to reach a consensus on the state of the metachain — a very slow process.

Metachains can, of course, be implemented on Themelio, but Themelio's more expressive features can greatly increase their performance. For example, all transactions claiming to be part of the metachain can be placed in a permissionless Catena log (see Section 3.4.3). Metachain clients could then avoid scanning through the mass of unrelated Themelio transactions.

If the state transition function of the metachain can be expressed in a short Melodeon constraint, Themelio could even enforce the rules of the metachain. This can be combined with using a unique "non-fungible token" inside the Catena log to label the metachain. That way, any light client can request the transaction with the one and only unspent output denominated in that token, and that transaction is guaranteed to contain the latest state of the metachain.

As described, metachains would be public and permissionless, but similar techniques can be used to secure private blockchains. Data in metachains can be encrypted with a key that only authorized parties know, and the Catena log can have a signature-checking constraint to block unauthorized users from spamming the metachain. Permissioned metachains have a very attractive combination — they inherit the immutability and trustlessness of Themelio, while preventing public access to the contents of the metachain.

8.4 A decentralized, secure Internet

Finally, let us envision an Internet with strong endogenous trust for critical applications and ubiquitous decentralization. This is similar in spirit to the "Web3" concept of decentralized network applications promoted by platform blockchains like Ethereum.

Compared to one based on platform blockchains, which require a large amount of on-chain software engineering, as well as deployment of heavyweight blockchain software to avoid pervasive trust assumptions on API "gateways" that host trusted views to on-chain state, a Themelio-based "web3" would be much more amenable to gradual evolution from existing network applications. This is because due to the easily embeddable nature of Themelio's endogenous trust, endogenous trust can be applied to the most security-critical parts of the Internet and gradually extended to the rest.

For instance, we can imagine a Facebook-like social network where a centralized provider is still used to publish everyone's posts, but posts are also replicated to decentralized CDNs like IPFS, private communication is protected by a Bitforest PKI, and users can "tip" each other through Astrape payment channels. In this hypothetical system, almost no trust is placed on the provider except for guaranteeing availability. Censorship resistance, confidentiality, and payment security are all guaranteed by the Themelio blockchain's endogenous trust, mediated through protocols like Astrape and Bitforest.

Finally, we can envision network applications where all security properties are fully based upon endogenous trust. For example, a BitChute-like video hosting service might involve an immutable autonomous application issuing tokens to incentivize a decentralized CDN, in the style of FileCoin, making availability fully endogenous. Cryptographic commitments to indices of all available content, analogous to .torrent files in BitTorrent, can be committed onto a Catena chain in Themelio with an appropriate covenant, ensuring global consistency and naming without any trusted party. Such a "ThemelioChute" video distribution service would be mostly trustless, scalable, and off-chain, while bearing little resemblance to the directly blockchain-backed applications of "Web3".

Chapter 9

Conclusion

Public blockchains, as originally envisioned, herald a fundamental revolution in the way trust works in distributed systems. Unfortunately, they have not seen widespread usage in production systems, outside of a few applications using private blockchains that eschew most of blockchains' distinctiveness. Blockchain development has also run into many serious obstacles, such as scalability and governance, that have thus far evaded satisfactory solutions.

We propose that instead of technical limitations, the key obstacle to blockchains' success is poor implementation of their most important property — endogenous trust. Through a survey of existing blockchains, we conclude that this lack of endogenous trust is due to a combination of weak cryptoeconomic design and an incorrect layering paradigm — current blockchains are generally too close to the application layer, forcing complex blockchain implementations on one hand and "leaky", rigid applications on the other hand. We propose that the correct paradigm for blockchains is that of a minimal root of trust, providing the "secret sauce" of endogenous trust to applications that mostly run outside the blockchain.

We described Themelio, a blockchain we developed to support this vision, using many novel technologies and design tradeoffs not seen in current blockchains. Themelio focuses on stability and security in three main aspects — a simple yet highly composable coin-based data model, a cryptoeconomically robust proof of stake system (Synkletos) based on simulating a unified entity, and finally a trustless stablecoin (mel) that stabilizes the value of Themelio's base currency without compromising endogenous trust.

Finally, we demonstrated the wide range of applications that can be developed using Themelio within a blockchain-minimizing paradigm. Two protocols we developed — Astrape, an anonymous payment channel construction, and Bitforest, a trustless naming system — were used as case studies of mid-level protocols that allow applications to access Themelio's endogenous trust despite

Themelio's simple and conservative protocol. We also sketched many examples of applications, from low-level to abstract, that may use mid-level protocols and Themelio to attain high performance, flexibility, and strong endogenous trust.

Bibliography

[19] *Lightning Network Daemon*. https://github.com/lightningnetwork/lnd. 2019.

[ACD17] Nader Al-Naji, Josh Chen, and Lawrence Diao. *Basis: A Price-Stable Cryptocurrency with an Algorithmic Central Bank*. 2017.

[Al-18] Nader Al-Naji. *Basis update*. 2018. URL: https://medium.com/basis-blog/basis-update-ae96e3565b1d (visited on 09/15/2019).

[Ali+16] Muneeb Ali, Jude Nelson, Ryan Shea, and Michael J Freedman. "Blockstack: A global naming and storage system secured by blockchains". In: *2016 USENIX annual technical conference (USENIX ATC 16)*. 2016, pp. 181–194.

[Ali+19] Muhammad Salek Ali, Massimo Vecchio, Miguel Pincheira, Koustabh Dolui, Fabio Antonelli, and Mubashir Husain Rehmani. "Applications of Blockchains in the Internet of Things: A Comprehensive Survey". In: *IEEE Communications Surveys & Tutorials* 21.2 (2019), pp. 1676–1717. DOI: 10.1109/COMST.2018.2886932.

[AMM21] Lukas Aumayr, Pedro Monero-Sanchez, and Matteo Maffei. "Blitz: Secure Multi-Hop Payments Without Two-Phase Commits". In: *30th USENIX Security Symposium*. 2021.

[Ang+19] Guillermo Angeris, Hsien-Tang Kao, Rei Chiang, Charlie Noyes, and Tarun Chitra. "An analysis of Uniswap markets". In: *arXiv preprint arXiv:1911.03380* (2019).

[Aum+13] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. "BLAKE2: simpler, smaller, fast as MD5". In: *International Conference on Applied Cryptography and Network Security*. Springer. 2013, pp. 119–135.

[Aum+21] Lukas Aumayr, Matteo Maffei, Oğuzhan Ersoy, Andreas Erwig, Sebastian Faust, Siavash Riahi, Kristina Hostáková, and Pedro Moreno-Sanchez. "Bitcoin-compatible virtual channels". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 901–918.

[Bac+13] Michael Backes, Aniket Kate, Praveen Manoharan, Sebastian Meiser, and Esfandiar Mohammadi. "AnoA: A framework for analyzing anonymous communication protocols". In: *2013 IEEE 26th Computer Security Foundations Symposium*. IEEE. 2013, pp. 163–178.

[Bis] Bisq. *A decentralized Bitcoin Exchange Network*. URL: https://bisq.network/.

[Bit19a] BitPay. *BitPay*. 2019. URL: https://bitpay.com (visited on 10/16/2019).

[Bit19b] BitPremier. *The Bitcoin Volatility Index*. 2019. URL: https://bitvol.info/ (visited on 10/01/2019).

[Blo18] Ludong BlockBeats. *The EOS supernode election: a national struggle worth "hundreds of billions" (in Chinese)*. 2018. URL: https://zhuanlan.zhihu.com/p/34902188 (visited on 04/01/2019).

[Blo21] Blockchain.com. *Blockchain Charts*. https://www.blockchain.com/charts. 2021. (Visited on 04/01/2022).

[Bon16] Joseph Bonneau. "EthIKS: Using Ethereum to audit a CONIKS key transparency log". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 95–105.

[Bri+23] Antonio Briola, David Vidal-Tomás, Yuanrong Wang, and Tomaso Aste. "Anatomy of a Stablecoin's failure: The Terra-Luna case". In: *Finance Research Letters* 51 (2023), p. 103358.

[But+14a] Vitalik Buterin et al. "Ethereum white paper, 2014". In: (2014). URL: https://github.com/ethereum/wiki/wiki/White-Paper.

[But+14b] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. "EIP-1559: Fee market change for ETH 1.0 chain". In: (2014). URL: https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md.

[But14] Vitalik Buterin. *The search for a stable cryptocurrency*. Nov. 11, 2014. URL: https://blog.ethereum.org/2014/11/11/search-stable-cryptocurrency/ (visited on 08/15/2019).

[But18a] Vitalik Buterin. *DRAFT: Position paper on resource pricing*. 2018. URL: https://ethresear.ch/t/draft-position-paper-on-resource-pricing (visited on 10/03/2019).

[But18b] Vitalik Buterin. *Estimating cryptocurrency demand elasticity from natural experiments*. 2018. URL: https://ethresear.ch/t/estimating-cryptocurrency-transaction-demand-elasticity-from-natural-experiments/2330.

[But19a] Vitalik Buterin. *Base Layers And Functionality Escape Velocity*. 2019. URL: https://vitalik.ca/general/2019/12/26/mvb.html.

[But19b] Vitalik Buterin. *Ethereum Proof of Stake FAQ*. 2019. URL: https://github.com/ethereum/wiki/wiki/Proof-of-Stake-FAQ (visited on 03/29/2019).

[But19c] Vitalik Buterin. *On Collusion*. 2019. URL: https://vitalik.ca/general/2019/04/03/collusion.html.

[But21] Vitalik Buterin. *Why sharding is great: demystifying the technical properties*. 2021. URL: https://vitalik.ca/general/2021/04/07/sharding.html.

[Cac16] Christian Cachin. "Architecture of the Hyperledger blockchain fabric". In: *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*. 2016.

[Cam+18] Jan Camenisch, Manu Drijvers, Tommaso Gagliardoni, Anja Lehmann, and Gregory Neven. "The wonderful world of global random oracles". In: *IACR EuroCrypt*. Springer. 2018, pp. 280–312.

[Car+16] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. "On the instability of bitcoin without the block reward". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 154–167.

[Cha22] Chainalysis. *The Trades That Triggered TerraUSD's Collapse*. Blog post. 2022. URL: https://blog.chainalysis.com/reports/how-terrausd-collapsed/.

[Chi+19] Tarun Chitra, Monica Quaintance, Stuart Haber, and Will Martino. "Agent-based simulations of blockchain protocols illustrated via kadena's chainweb". In: *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE. 2019, pp. 386–395.

[Coi] CoinMarketCap. *Cryptocurrency Market Capitalizations*. URL: https://coinmarketcap.com (visited on 02/14/2022).

[Coi22] Cointelegraph. *Alchemy and Infura block access to Tornado Cash as Vitalik Buterin weighs in on debate*. https://cointelegraph.com/news/alchemy-and-infura-block-access-to-tornado-cash-as-vitalik-buterin-weighs-in-on-debate. 2022. (Visited on 06/30/2023).

[CP18] Bram Cohen and Krzysztof Pietrzak. "Simple proofs of sequential work". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 451–467.

[cra21] Rust crate. *bincode encoding, Rust crate*. 2021. URL: https://crates.io/crates/bincode.

[Cro+16] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, et al. "On scaling decentralized blockchains". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 106–125.

[cry19] crypto51. *PoW 51% Attack Cost*. 2019. URL: https://www.crypto51.app/.

[CS20] Benjamin Y Chan and Elaine Shi. "Streamlet: Textbook streamlined blockchains". In: *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 2020, pp. 1–11.

[DB19] Yuhao Dong and Raouf Boutaba. "Elasticoin: Low-Volatility Cryptocurrency with Proofs of Sequential Work". In: *IEEE International Conference on Blockchain and Cryptocurrency*. IEEE. IEEE, 2019, pp. 205–209.

[Del+14] Antoine Delignat-Lavaud, Martın Abadi, Andrew Birrell, Ilya Mironov, Ted Wobber, and Yinglian Xie. "Web PKI: Closing the Gap between Guidelines and Practices." In: *NDSS*. 2014.

[DG09] George Danezis and Ian Goldberg. "Sphinx: A compact and provably secure mix format". In: *30th IEEE Symposium on Security and Privacy*. IEEE. 2009, pp. 269–282.

[DHK21] Claudia Diaz, Harry Halpin, and Aggelos Kiayias. "The Nym Network". In: *(self-published)* (2021).

[DKB18] Yuhao Dong, Woojung Kim, and Raouf Boutaba. "Bitforest: a Portable and Efficient Blockchain-Based Naming System". In: *2018 14th International Conference on Network and Service Management (CNSM)*. IEEE. 2018, pp. 226–232.

[DLS88] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. "Consensus in the presence of partial synchrony". In: *Journal of the ACM (JACM)* 35.2 (1988), pp. 288–323.

[DM98] Jean-Pierre Danthine and Serge Moresi. "Front-running by mutual fund managers: a mixed bag". In: *Review of Finance* 2.1 (1998), pp. 29–56.

[DMN18] Patrick Dai, Neil Mahi, and Alex Norta. "Smart-Contract Value-Transfer Protocols on a Distributed Mobile Application Platform". In: (2018). URL: https://whitepaper.io/document/8/qtum-whitepaper.

[DMS04] Roger Dingledine, Nick Mathewson, and Paul Syverson. *Tor: The second-generation onion router*. Tech. rep. Naval Research Lab Washington DC, 2004.

[DPP16] Rasmus Dahlberg, Tobias Pulls, and Roel Peeters. "Efficient sparse merkle trees". In: *Nordic Conference on Secure IT Systems*. Springer. 2016, pp. 199–215.

[DRO18] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. *eltoo: A Simple Layer2 Protocol for Bitcoin*. https://blockstream.com/eltoo.pdf. 2018.

[DW15] Christian Decker and Roger Wattenhofer. "A fast and scalable payment network with bitcoin duplex micropayment channels". In: *Symposium on Self-Stabilizing Systems*. Springer. 2015, pp. 3–18.

[E-E] E-Estonia. *KSI Blockchain*. URL: https://e-estonia.com/solutions/security-and-safety/ksi-blockchain/.

[Eng+17] Felix Engelmann, Henning Kopp, Frank Kargl, Florian Glaser, and Christof Weinhardt. "Towards an economic analysis of routing in payment channel networks". In: *Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*. 2017, pp. 1–6.

[EOS22] EOS. *EOS.IO*. Available: https://eos.io. 2022. URL: https://eos.io.

[ES14] Ittay Eyal and Emin Gün Sirer. "Majority is not enough: Bitcoin mining is vulnerable". In: *International conference on financial cryptography and data security*. Springer. 2014, pp. 436–454.

[Eth] Ethereum. *History of Ethereum*. URL: https://ethereum.org/en/history (visited on 06/28/2023).

[Feu84] Albert Feuerwerker. "The state and the economy in late imperial China". In: *Theory and Society* 13.3 (1984), pp. 297–326.

[FJS10] Joan Feigenbaum, Aaron Johnson, and Paul Syverson. "Preventing active timing attacks in low-latency anonymous communication". In: *International Symposium on Privacy Enhancing Technologies Symposium*. Springer. 2010, pp. 166–183.

[Flo18] David Floyd. *EOS' Blockchain Arbitrator Orders Freeze of 27 Accounts*. 2018. URL: https://www.coindesk.com/eos-blockchain-arbitrator-orders-freeze-of-27-accounts.

[Fou22] IOTA Foundation. *Coordinator*. 2022. URL: https://wiki.iota.org/learn/about-iota/coordinator (visited on 07/05/2023).

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, pp. 186–194.

[FVY14] Conner Fromknecht, Dragos Velicanu, and Sophia Yakoubov. "A Decentralized Public Key Infrastructure with Identity Retention." In: *IACR Cryptol. ePrint Arch.* 2014 (2014), p. 803.

[Gil+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. "Algorand: Scaling byzantine agreements for cryptocurrencies". In: *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM. 2017, pp. 51–68.

[GM17] Matthew Green and Ian Miers. "Bolt: Anonymous payment channels for decentralized currencies". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 473–489.

[GS18] John M Griffin and Amin Shams. "Is bitcoin really un-tethered?" In: *Available at SSRN 3195066* (2018).

[Haj21] Nermin Hajdarbegovic. *Bitcoin miners ditch Ghash.io pool over fears of 51% attack*. Sept. 2021. URL: https://www.coindesk.com/markets/2014/01/09/bitcoin-miners-ditch-ghashio-pool-over-fears-of-51-attack/.

[Hei+15] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. "Eclipse attacks on Bitcoin's peer-to-peer network". In: *24th USENIX security symposium (USENIX security 15)*. 2015, pp. 129–144.

[Hei+17] Ethan Heilman, Leen Alshenibr, Foteini Baldimtsi, Alessandra Scafuro, and Sharon Goldberg. "Tumblebit: An untrusted bitcoin-compatible anonymous payment hub". In: *Network and Distributed System Security Symposium*. 2017.

[HLS18] Thorsten Hens, Terje Lensberg, and Klaus Reiner Schenk-Hoppé. "Front-Running and Market Quality: An Evolutionary Perspective on High Frequency Trading". In: *International Review of Finance* 18.4 (2018), pp. 727–741.

[HM02] John F Hornbeck and Meaghan K Marshal. "The Argentine financial crisis: a chronology of events". In: Congressional Research Service [Library of Congress]. 2002.

[Inf23] Infura. *Infura: Blockchain Infrastructure for the New Internet*. https://infura.io/. 2023. (Visited on 06/30/2023).

[Iwa+14] Mitsuru Iwamura, Yukinobu Kitamura, Tsutomu Matsumoto, and Kenji Saito. "Can we stabilize the price of a Cryptocurrency?: Understanding the design of Bitcoin and its potential to compete with Central Bank money". In: *Understanding the Design of Bitcoin and Its Potential to Compete with Central Bank Money (October 25, 2014)* (2014).

[Jev85] William Stanley Jevons. *Money and the Mechanism of Exchange*. Vol. 17. Kegan Paul, Trench, 1885.

[JL17] Simon Josefsson and Ilari Liusvaara. *Edwards-Curve Digital Signature Algorithm (EdDSA)*. RFC 8032. Jan. 2017. DOI: 10.17487/RFC8032. URL: https://rfc-editor.org/rfc/rfc8032.txt.

[JMU22] Akanksha Jalan, Roman Matkovskyy, and Andrew Urquhart. "Demand elasticities of Bitcoin and Ethereum". In: *Economics Letters* 220 (2022), p. 110877.

[Kal+15] Harry A Kalodner, Miles Carlsten, Paul Ellenbogen, Joseph Bonneau, and Arvind Narayanan. "An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design." In: *WEIS*. Citeseer. 2015.

[Kam11] Dan Kaminsky. *Spelunking the triangle: Exploring Aaron Swartz's take on Zooko's triangle*. 2011.

[Lai+17] Russell W. F. Lai, Henry K. F. Cheung, Sherman S. M. Chow, and Anthony Man-Cho So. "Another Look at Anonymous Communication". In: *Paradigms in Cryptology – Mycrypt 2016. Malicious and Exploratory Cryptology*. Ed. by Raphaël C.-W. Phan and Moti Yung. Cham: Springer International Publishing, 2017, pp. 56–82. ISBN: 978-3-319-61273-7.

[Leu+18] Derek Leung, Adam Suhl, Yossi Gilad, and Nickolai Zeldovich. "Vault: Fast bootstrapping for the algorand cryptocurrency". In: *Cryptology ePrint Archive* (2018).

[Li+20] Xiaoqi Li, Peng Jiang, Ting Chen, Xiapu Luo, and Qiaoyan Wen. "A survey on the security of blockchain systems". In: *Future Generation Computer Systems* 107 (2020), pp. 841–853.

[LL17] Iuon-Chang Lin and Tzu-Chun Liao. "A survey of blockchain security issues and challenges." In: *Int. J. Netw. Secur.* 19.5 (2017), pp. 653–659.

[Mal+17] Giulio Malavolta, Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Srivatsan Ravi. "Concurrency and privacy with payment-channel networks". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 455–471.

[Mal+19] Giulio Malavolta, Pedro Moreno-Sanchez, Clara Schneidewind, Aniket Kate, and Matteo Maffei. "Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability." In: *NDSS*. 2019.

[McC+16] Patrick McCorry, Malte Möser, Siamak F Shahandasti, and Feng Hao. "Towards bitcoin payment networks". In: *Australasian Conference on Information Security and Privacy*. Springer. 2016, pp. 57–76.

[Mel+15] Marcela S Melara, Aaron Blankstein, Joseph Bonneau, Edward W Felten, and Michael J Freedman. "CONIKS: Bringing Key Transparency to End Users". In: *24th USENIX Security Symposium (USENIX Security '15)*. 2015, pp. 383–398.

[Mel23] Melscan. *Melscan*. 2023. URL: `https://melscan.io` (visited on 07/05/2023).

[MJ15] N Moeller and S Josefsson. *IETF draft: EdDSA and Ed25519*. 2015.

[Mor17] Daniel Morgan. *The Great Bitcoin Scaling Debate — A Timeline*. 2017. URL: `https://medium.com/hackernoon/the-great-bitcoin-scaling-debate-a-timeline-6108081dbada` (visited on 08/01/2020).

[MRV99] Silvio Micali, Michael Rabin, and Salil Vadhan. "Verifiable random functions". In: *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE. 1999, pp. 120–130.

[MSA19] Ahmed Afif Monrat, Olov Schelén, and Karl Andersson. "A survey of blockchain from the perspectives of applications, challenges, and opportunities". In: *IEEE Access* 7 (2019), pp. 117134–117151.

[Mur15] Steven J. Murdoch. *How Tor's privacy was (momentarily) broken, and the questions it raises*. 2015. URL: https://theconversation.com/how-tors-privacy-was-momentarily-broken-and-the-questions-it-raises-52048 (visited on 04/01/2022).

[Nak08] Satoshi Nakamoto. "Bitcoin: A peer-to-peer electronic cash system". In: (2008).

[Nam] Namecoin. *Namecoin*. URL: https://www.namecoin.org (visited on 06/29/2023).

[NB13] André Niemann and Jacqueline Brendel. *A survey on CA compromises*. 2013.

[OCo+20] Jack O'Conner, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. *BLAKE3: One function, fast everywhere*. 2020.

[OP17] Russell O'Connor and Marta Piekarska. "Enhancing Bitcoin transactions with covenants". In: *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer. 2017, pp. 191–198.

[PC11] J Ronald Prins and Business Unit Cybercrime. "Diginotar certificate authority breach 'operation black tulip'". In: *Fox-IT, November* (2011).

[PH10] Andreas Pfitzmann and Marit Hansen. *A terminology for talking about privacy by data minimization: Anonymity, unlinkability, undetectability, unobservability, pseudonymity, and identity management*. 2010. URL: http://www.maroki.de/pub/dphistory/2010_Anon_Terminology_v0.34.pdf (visited on 08/01/2023).

[PP17] Alessandra Prentice and Pavel Polityuk. "Global cyber attack likely cover for covert malware installation: Ukraine police official". In: *Reuters* (July 2017). URL: https://www.reuters.com/article/cyber-attack-ukraine/global-cyber-attack-likely-cover-for-covert-malware-installation-ukraine-police-official-idUSL8N1JQ4DG.

[RN17] Ronny Richardson and Max North. "Ransomware: Evolution, mitigation and prevention". In: *International Management Review* 13.1 (2017), pp. 10–21.

[Sam19] Robert Sams. *A Note on Cryptocurrency Stabilisation: Seigniorage Shares*. Aug. 27, 2019. URL: https://github.com/rmsams/stablecoins.

[Sec] SecureKey. *Building Trusted Identity Networks*. URL: https://securekey.com/.

[Ser23] Ethereum Name Service. *Ethereum Name Service*. 2023. URL: https://ens.domains/ (visited on 06/18/2023).

[Sol23] Solana. *Solana: Web3 infrastructure for everyone*. 2023. URL: https://solana.com (visited on 07/01/2023).

[SSZ16] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. "Optimal selfish mining strategies in bitcoin". In: *International Conference on Financial Cryptography and Data Security*. Springer. 2016, pp. 515–532.

[Sun18] Cass R Sunstein. *Republic: Divided democracy in the age of social media*. Princeton University Press, 2018.

[TD17] Alin Tomescu and Srinivas Devadas. "Catena: Efficient non-equivocation via bitcoin". In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 393–409.

[Tea17] Maker Team. "The Dai Stablecoin System". In: (2017). URL: https://makerdao.com/whitepaper/DaiDec17WP.pdf.

[Tet] Tether. *Tether*. URL: https://tether.io (visited on 10/16/2019).

[TMM21] Erkan Tairi, Pedro Moreno-Sanchez, and Matteo Maffei. "A2L: Anonymous atomic locks for scalability and interoperability in payment channel hubs". In: *42nd IEEE Symposium on Security and Privacy*. 2021.

[Tru] TrustToken. *TrueUSD*. URL: https://www.trusttoken.com/trueusd (visited on 10/03/2019).

[TT16] Don Tapscott and Alex Tapscott. *Blockchain revolution: how the technology behind bitcoin is changing money, business, and the world*. Penguin, 2016.

[Van16] Aaron Van Wirdum. *How the Lightning Network Layers Privacy on Top of Bitcoin*. https://bitcoinmagazine.com/articles/how-the-lightning-network-layers-privacy-on-top-of-bitcoin-1482183775. 2016. (Visited on 04/01/2022).

[Ver+18] Joannes Vermorel, Amaury Sechet, Shammah Chancellor, and T Wansem. *Canonical transaction ordering for bitcoin*. 2018.

[Wal89] Alan Walters. "Currency boards". In: *Money*. Springer, 1989, pp. 109–114.

[Wan17] Maya Wang. "China's Chilling 'Social Credit' Blacklist". In: *The Wall Street Journal* 11 (2017).

[WG18] Karl Wüst and Arthur Gervais. "Do you need a blockchain?" In: *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*. IEEE. 2018, pp. 45–54.

[Wik] Ethereum Wiki. *Hard problems of cryptocurrency*. URL: https://github.com/ethereum/wiki/wiki/Problems (visited on 08/15/2019).

[Woo14] Gavin Wood. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum Project Yellow Paper* 151.2014 (2014), pp. 1–32.

[Yag+18] Dylan Yaga, Peter Mell, Nik Roby, and Karen Scarfone. *Blockchain technology overview*. Tech. rep. Oct. 2018. DOI: 10.6028/nist.ir.8202. URL: https://doi.org/10.6028/nist.ir.8202.

[Yli+16] Jesse Yli-Huumo, Deokyoon Ko, Sujin Choi, Sooyong Park, and Kari Smolander. "Where is current research on blockchain technology?—a systematic review". In: *PloS one* 11.10 (2016), e0163477.

[You+21] H Yousaf, G Kappos, A Piotrowska, S Kanjalkar, S Delgado-Segura, A Miller, and S Meiklejohn. "An Empirical Analysis of Privacy in the Lightning Network". In: *Financial Cryptography and Data Security* (2021).

[Zhe+18] Zibin Zheng, Shaoan Xie, Hong-Ning Dai, Xiangping Chen, and Huaimin Wang. "Blockchain challenges and opportunities: A survey". In: *International Journal of Web and Grid Services* 14.4 (2018), pp. 352–375.