Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks

Lorenz Breidenbach[1] Christian Cachin[2] Benedict Chan[1]
Alex Coventry[1] Steve Ellis[1] Ari Juels[3] Farinaz Koushanfar[4]
Andrew Miller[5] Brendan Magauran[1] Daniel Moroz[6]
Sergey Nazarov[1] Alexandru Topliceanu[1] Florian Tramèr[7]
Fan Zhang[8]

15 April 2021
v1.0

[1]Chainlink Labs
[2]The author is a faculty member at University of Bern. He co-authored this work in his separate capacity as an advisor to Chainlink Labs.
[3]The author is a faculty member at Cornell Tech. He co-authored this work in his separate capacity as Chief Scientist at Chainlink Labs.
[4]The author is a faculty member at University of California, San Diego. She co-authored this work in her separate capacity as an advisor to Chainlink Labs.
[5]The author is a faculty member at University of Illinois Urbana-Champaign. He co-authored this work in his separate capacity as an advisor on Chainlink.
[6]The author is a PhD candidate at Harvard University. He co-authored this work while on leave from Harvard, in his capacity as a researcher at Chainlink Labs.
[7]The author is a PhD candidate at Stanford University. He co-authored this work in his separate capacity as an advisor to Chainlink Labs.
[8]The author will be joining the faculty of Duke University in fall 2021. He co-authored this work in his current capacity as a researcher at Chainlink Labs.

Abstract

In this whitepaper, we articulate a vision for the evolution of Chainlink beyond its initial conception in the original Chainlink whitepaper. We foresee an increasingly expansive role for oracle networks, one in which they complement and enhance existing and new blockchains by providing fast, reliable, and confidentiality-preserving universal connectivity and off-chain computation for smart contracts.

The foundation of our plan is what we call *Decentralized Oracle Networks*, or DONs for short. A DON is a network maintained by a committee of Chainlink nodes. It supports any of an unlimited range of oracle functions chosen for deployment by the committee. A DON thus acts as a powerful abstraction layer, offering interfaces for smart contracts to extensive off-chain resources and highly efficient yet decentralized off-chain computing resources within the DON itself.

With DONs as a springboard, Chainlink plans to focus on advances in seven key areas:

- *Hybrid smart contracts:* Offering a powerful, general framework for augmenting existing smart contract capabilities by securely composing on-chain and off-chain computing resources into what we call *hybrid smart contracts*.
- *Abstracting away complexity:* Presenting developers and users with simple functionality eliminates the need for familiarity with complex underlying protocols and system boundaries.
- *Scaling:* Ensuring that oracle services achieve the latencies and throughputs demanded by high-performance decentralized systems.
- *Confidentiality:* Enabling next-generation systems that combine blockchains' innate transparency with strong new confidentiality protections for sensitive data.
- *Order-fairness for transactions:* Supporting transaction sequencing in ways that are fair for end users and prevent front-running and other attacks by bots and exploitative miners.
- *Trust-minimization:* Creating a highly trustworthy layer of support for smart contracts and other oracle-dependent systems by means of decentralization, strong anchoring in high-security blockchains, cryptographic techniques, and cryptoeconomic guarantees.
- *Incentive-based (cryptoeconomic) security:* Rigorously designing and robustly deploying mechanisms that ensure nodes in DONs have strong economic incentives to behave reliably and correctly, even in the face of well-resourced adversaries.

We present preliminary and ongoing innovations by the Chainlink community in each of these areas, providing a picture of the broadening and increasingly powerful capabilities planned for the Chainlink network.

Contents

1 Introduction **6**
- 1.1 Decentralized Oracle Networks . . . 7
- 1.2 Seven Key Design Goals . . . 8
- 1.3 Organization of this Paper . . . 19

2 Security Model and Goals **19**
- 2.1 Current Architectural Model . . . 20
- 2.2 Consensus Assumptions . . . 20
- 2.3 Notation . . . 22
- 2.4 Note on Trust Models . . . 22

3 Decentralized Oracle Network Interface and Capabilities **23**
- 3.1 Networking . . . 24
- 3.2 Computation . . . 25
- 3.3 Storage . . . 27
- 3.4 Transaction-Execution Framework (TEF) . . . 27
- 3.5 Mempool Services . . . 28
- 3.6 Stepping Stones: Existing Chainlink Capabilities . . . 29
  - 3.6.1 Off-Chain Reporting (OCR) . . . 29
  - 3.6.2 DECO and Town Crier . . . 30
  - 3.6.3 Existing On-Chain Chainlink Services . . . 31
  - 3.6.4 Node Reputation / Performance History . . . 33

4 Decentralized Services Enabled by Decentralized Oracle Networks **34**
- 4.1 Proof of Reserves . . . 35
- 4.2 Interfacing with Enterprise / Legacy Systems . . . 35
- 4.3 Decentralized Identity . . . 36
- 4.4 Priority Channels . . . 39
- 4.5 Confidentiality-Preserving DeFi / Mixicles . . . 41

5 Fair Sequencing Services **43**
- 5.1 The Front-Running Problem . . . 45
  - 5.1.1 Oracle Front-Running . . . 45
  - 5.1.2 Front-Running User Transactions . . . 47
- 5.2 FSS Details . . . 48
  - 5.2.1 Transaction Processing . . . 48
  - 5.2.2 Transaction Atomicity . . . 51
- 5.3 Fair Transaction Sequencing . . . 53
- 5.4 Network-Layer Considerations . . . 56
- 5.5 Entity-Level Fairness Policies . . . 57

6 The DON Transaction-Execution Framework (DON-TEF) 57
6.1 TEF Overview . . . 58
6.2 Transaction Routing . . . 60
6.3 Syncing . . . 61
6.4 Reorgs . . . 64

7 Trust Minimization 65
7.1 Data-Source Authentication . . . 65
7.1.1 The Limitations of Authenticated Data Origination . . . 67
7.1.2 Confidentiality . . . 67
7.1.3 Combining Source Data . . . 68
7.1.4 Processing Source Data . . . 69
7.2 DON Trust Minimization . . . 70
7.2.1 Failover Clients . . . 70
7.2.2 Minority Reports . . . 70
7.3 Guard Rails . . . 71
7.4 Trust-Minimized Governance . . . 72
7.5 Public-Key Infrastructure . . . 73

8 DON Deployment Considerations 75
8.1 Rollout Approach . . . 76
8.2 Dynamic DON Membership . . . 76
8.3 DON Accountability . . . 77

9 Economics and Cryptoeconomics 78
9.1 Staking Overview . . . 81
9.2 Background . . . 83
9.3 Modeling Assumptions . . . 84
9.3.1 First-Tier Incentive Model: Rational Actors . . . 85
9.3.2 Second-Tier Adjudication Model: Correctness by Assumption . . . 85
9.3.3 Adversarial Model . . . 86
9.3.4 How Much Cryptoeconomic Security Is Enough? . . . 87
9.4 Staking Mechanism: Sketch . . . 87
9.4.1 Further Mechanism Details . . . 89
9.4.2 Quadratic Staking Impact . . . 90
9.4.3 Realization of Second Tier . . . 91
9.4.4 Misreporting Insurance . . . 93
9.5 Single-Round Variant . . . 94
9.6 Implicit-Incentive Framework (IIF) . . . 95
9.6.1 Future Fee Opportunity . . . 96
9.6.2 Speculative FFO . . . 97
9.6.3 External Reputation . . . 98

9.6.4 Open IIF Analytics . . . 98
9.7 Putting It All Together: Node Operator Incentives . . . 99
9.8 The Virtuous Cycle of Economic Security . . . 100
9.9 Additional Factors Driving Network Growth . . . 101

10 Conclusion **102**

A Glossary **116**

B DON Interface: Further Details **119**
B.1 Networking . . . 119
B.1.1 Integrity . . . 120
B.1.2 Confidentiality . . . 121
B.1.3 Availability . . . 122
B.2 Computation . . . 122
B.2.1 Trusted Execution Environments (TEEs) . . . 122
B.2.2 TEE Security . . . 123
B.2.3 Secure Multi-Party Computation (MPC) . . . 124
B.3 Storage . . . 125
B.4 Resource Pricing . . . 125

C Adapter Examples **126**
C.1 Oracle-Mediated Data-Source Access (MediatedReport) . . . 126
C.2 Cross-Ledger Reports (XL-Report-Read) . . . 127
C.3 Confidential Switch (ConfSwitch) . . . 128

D Functional Signatures **129**
D.1 Functional Signatures for Combining Data . . . 130
D.2 Discretized Functional Signatures . . . 130

E Prospective Bribery **134**

F Random vs. Committee-Based Oracle Selection **135**

1 Introduction

Blockchain *oracles* are often viewed today as decentralized services with one objective: to forward data from off-chain resources onto blockchains. It's a short step, though, from forwarding data to computing on it, storing it, or transmitting it bidirectionally. This observation justifies a much broader notion of oracles' functionality. So too do the growing service requirements of smart contracts and increasingly multifaceted technologies that rely on oracle networks. In short, an oracle can and will need to be a general-purpose, bidirectional, compute-enabled interface between and among on-chain and off-chain systems. Oracles' role in the blockchain ecosystem is to enhance the performance, functionality, and interoperability of smart contracts so that they can bring new trust models and transparency to a multiplicity of industries. This transformation will come about through broadening use of *hybrid smart contracts*, which fuse blockchains' special properties with the unique capabilities of off-chain systems such as oracle networks and thereby achieve far greater reach and power than on-chain systems in isolation.

In this whitepaper, we articulate a vision for what we call Chainlink 2.0, an evolution of Chainlink beyond its initial conception in the original Chainlink whitepaper [98]. We foresee an increasingly expansive role for oracle networks, one in which they complement and enhance existing and new blockchains by providing fast, reliable, and confidentiality-preserving universal connectivity and computation for hybrid smart contracts. We believe that oracle networks will even evolve to become utilities for exporting high-integrity blockchain-grade data to systems beyond the blockchain ecosystem.

Today, Chainlink nodes run by a diverse set of entities come together in oracle networks to relay data to smart contracts in what are known as *reports*. We can view such oracle nodes as a *committee* similar to that in a classical-consensus blockchain [72], but with the goal of supporting existing blockchains, rather than providing freestanding functionality. With verifiable random functions (VRF) and Off-Chain Reporting (OCR), Chainlink is already evolving toward a general-purpose framework and infrastructure for providing the computational resources that smart contracts require for advanced functionality.

The foundation of our plan for Chainlink 2.0 is what we call *Decentralized Oracle Networks*, or DONs for short. Since we introduced the term "oracle network" in the original Chainlink whitepaper [98], oracles have developed ever richer functionality and breadth of application. In this paper, we offer a fresh definition of the term according to our future vision for the Chainlink ecosystem. In this view, a DON is a network maintained by a committee of Chainlink nodes. Rooted in a consensus protocol, it supports any of an unlimited range of oracle functions chosen for deployment by the committee. A DON thus acts as a blockchain abstraction layer, providing interfaces to off-chain resources for both smart contracts and other systems. It also provides access to highly efficient yet decentralized off-chain computing resources. In general, a DON *supports operations on a main chain*. Its goal is to enable secure and flexi-

ble hybrid smart contracts, which combine on-chain and off-chain computation with connection to external resources.

We emphasize that even with the use of committees in DONs, Chainlink itself remains inherently permissionless. DONs act as the foundation of a permissionless framework in which nodes can come together to implement custom oracle networks with their own regimes for node inclusion, which may be permissioned or permissionless.

With DONs as a foundation, we plan to focus in Chainlink 2.0 on advances in seven key areas: *hybrid smart contracts*, *abstracting away complexity*, *scaling*, *confidentiality*, *order-fairness* for transactions, *trust minimization*, and *incentive-based (cryptoeconomic) security*. In this paper introduction, we present an overview of Decentralized Oracle Networks in Section 1.1 and then our seven key areas of innovation in Section 1.2. We describe the organization of the rest of this paper in Section 1.3.

1.1 Decentralized Oracle Networks

Decentralized Oracle Networks are designed to *enhance* and *extend* the capabilities of smart contracts on a *target blockchain* or *main chain* through functions that are not available natively. They do so by providing the three basic resources found in computing systems: networking, storage, and computation. A DON aims to offer these resources with strong confidentiality, integrity, and availability properties,[1] as well as accountability.

DONs are formed by committees of oracle nodes that cooperate to fulfill a specific job or choose to establish a long-lived relationship in order to provide persistent services to clients. DONs are designed in a blockchain-agnostic way. They promise to serve as a powerful and flexible tool for application developers to create off-chain support for their smart contracts on any supported main chain.

Two types of functionalities realize the capabilities of a DON: *executables* and *adapters*. *Executables* are programs that run continuously and in a decentralized manner on the DON. While they do not directly store main-chain assets, they have important benefits, including high performance and the ability to perform confidential computation. Executables run autonomously on a DON and perform deterministic operations. They work in hand with *adapters* that link the DON to external resources and may be called by executables. Adapters, as we envision them for DONs, are a generalization of the external adapters in Chainlink today. While existing adapters typically only fetch data from data sources, adapters may operate bidirectionally; in DONs, they may additionally leverage joint computation by DON nodes to achieve additional features, such as encrypting reports for privacy-preserving consumption by an executable.

To provide a sense of a DON's basic operation, Fig. 1 shows conceptually how a DON might be used to send reports to a blockchain and thus achieve traditional, existing oracle functionality. DONs can provide many additional features, however, beyond

[1] The "CIA triad" of information security [123, p. 26, §2.3.5].

Chainlink's existing networks. For example, within the general structure of Fig. 1, the executable could record fetched asset-price data on the DON, using such data to compute, e.g., a trailing average for its reports.

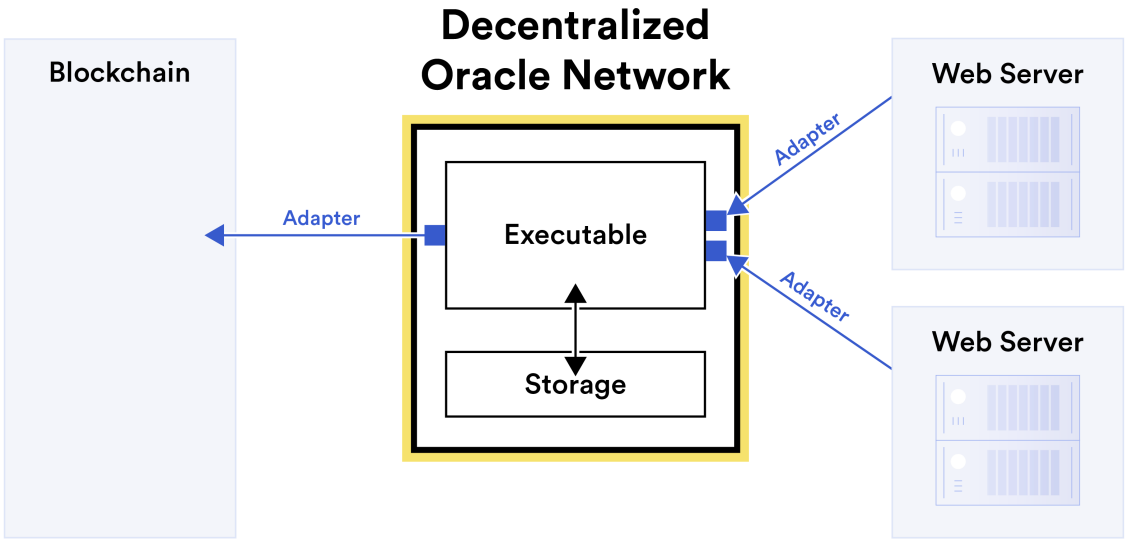

Figure 1: Conceptual figure showing as an example how a Decentralized Oracle Network can realize basic oracle functionality, i.e., relay off-chain data to a contract. An executable uses adapters to fetch off-chain data, which it computes on, sending output over another adapter to a target blockchain. (Adapters are initiated by code in the DON, represented by small blue boxes; arrows show the direction of data flow for this particular example.) The executable can additionally read and write to local DON storage to keep state and/or communicate with other executables. Flexible networking, computation, and storage in DONs, all represented here, enable a host of novel applications.

A major benefit of DONs is their ability to *bootstrap* new blockchain services. DONs are a vehicle by which existing oracle networks can quickly stand up service applications that would today require the creation of purpose-built networks. We give a number of examples of such applications in Section 4.

In Section 3, we provide more details on DONs, describing their capabilities in terms of the interface they present to developers and users.

1.2 Seven Key Design Goals

Here we briefly review the seven key focuses enumerated above for the evolution of Chainlink, namely:

Hybrid smart contracts: Central to our vision for Chainlink is the idea of securely combining on-chain and off-chain components in smart contracts. We refer to contracts realizing this idea as *hybrid smart contracts* or *hybrid contracts*.[2]

Blockchains are and will continue to play two critical roles in decentralized-service ecosystems: They are both the loci where cryptocurrency ownership is represented and robust anchors for decentralized services. Smart contracts must therefore be represented or executed on chain, but their on-chain capabilities are severely limited. Purely on-chain contract code is slow, expensive, and insular, unable to benefit from real-world data and a variety of functionalities that are inherently unachievable on chain, including various forms of confidential computation, generation of (pseudo)randomness secure against miner / validator manipulation, etc.

For smart contracts to realize their full potential therefore requires smart contracts to be architected with two parts: an on-chain part (which we typically denote by SC) and an off-chain part, an executable running on a DON (which we typically denote by exec). The goal is to achieve a secure composition of on-chain functionality with the multiplicity of off-chain services that DONs aim to provide. Together, the two parts make up a hybrid contract. We present the idea conceptually in Fig. 2. Already today, Chainlink services[3] such as data feeds and VRFs are enabling otherwise unachievable smart contract applications, ranging from DeFi to fairly generated NFTs to decentralized insurance, as first steps toward a more general framework. As Chainlink services expand and grow more performant according to our vision in this whitepaper, so too will the power of smart contract systems across all blockchains.

Our other six key focuses in this whitepaper may be viewed as acting in the service of the first, overarching one of hybrid contracts. These focuses involve removing visible complexity from hybrid contracts, creating additional off-chain services that enable the construction of ever more capable hybrid contracts, and, in the case of *trust minimization*, bolstering the security properties achieved by hybrid contracts. We leave the idea of hybrid contracts implicit throughout much of the paper, but any combination of MAINCHAIN logic with a DON may be viewed as a hybrid contract.

Abstracting away complexity: DONs are designed to make use of decentralized systems easy for developers and users by abstracting away the often complex machinery behind DONs' powerful and flexible array of services. Existing Chainlink services already have this feature. For example, data feeds in Chainlink today present on-chain interfaces that do not require developers to concern themselves with protocol-level details, such as the means by which OCR enforces consensus reporting among a

[2]The idea of on-chain / off-chain contract composition has arisen previously in various constrained forms, e.g., layer-2 systems, TEE-based blockchains [80], etc. Our goal is to support and generalize these approaches and ensure that they can encompass off-chain data access and other key oracle services.

[3]Chainlink services comprise a variety of decentralized services and functionality available through the network. They are offered by the numerous node operators composed into various oracle networks across the ecosystem.

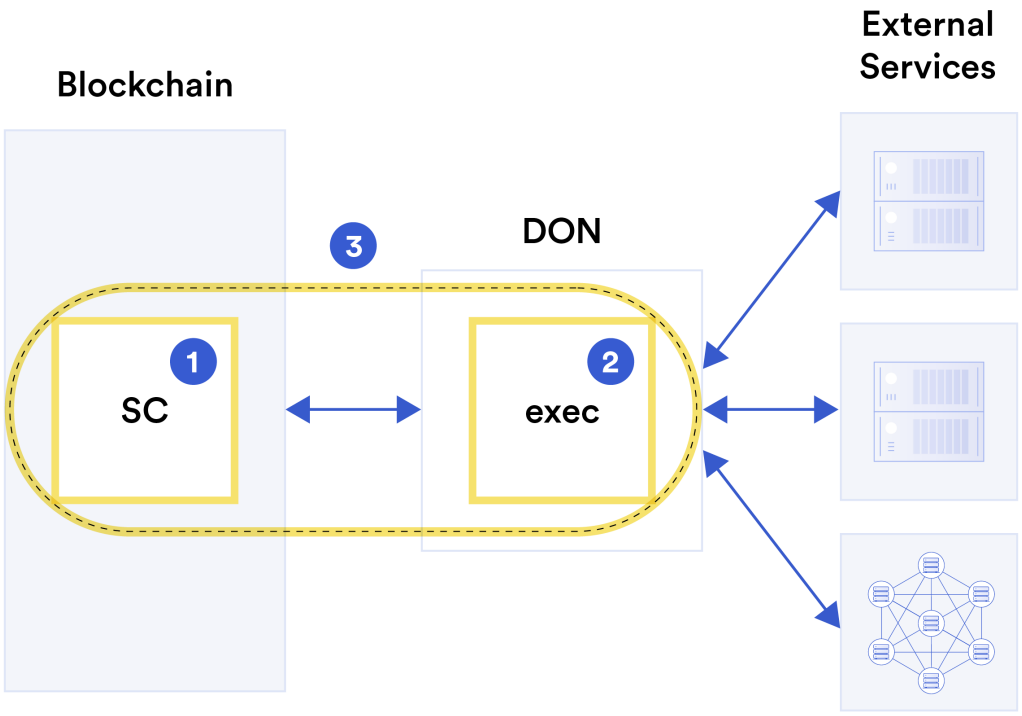

Figure 2: Conceptual figure depicting on-chain / off-chain contract composition. A hybrid smart contract ③ consists of two complementary components: an on-chain component SC ①, resident on a blockchain, and an off-chain component exec ② that executes on a DON. The DON serves as a bridge between the two components as well as connecting the hybrid contract with off-chain resources such as web services, other blockchains, decentralized storage, etc.

decentralized set of nodes. DONs go a step further in the sense that they expand the range of services for which Chainlink can offer developers an abstraction layer with accompanying streamlined interfaces for high-level services.

We present several application examples in Section 4 that highlight this approach. We envision enterprises, for instance, using DONs as a form of secure middleware to connect their legacy systems to blockchains. (See Section 4.2.) This use of DONs abstracts away the complexity of general blockchain dynamics (fees, reorgs, etc.). It also abstracts away the features of specific blockchains, thereby enabling enterprises to connect their existing systems to an ever-broadening array of blockchain systems without a need for specialized expertise in these systems or, more generally, in decentralized-systems development.

Ultimately, our ambition is to push the degree of abstraction achieved by Chainlink to the point of implementing what we refer to as a *decentralized metalayer*. Such a layer would abstract away the on-chain / off-chain distinction for all classes of developers and users of DApps, allowing seamless creation and use of decentralized services.

To simplify the development process, developers could specify DApp functionality in the metalayer as a virtual application in a unified machine model. They could then use a decentralized-metalayer compiler to instantiate the DApp automatically as a set of interoperating decentralized functionalities spanning blockchains, DONs, and external services. (One of these external services could be an enterprise system, making the metalayer useful for applications involving legacy enterprise systems.) Such compilation is akin to how modern compilers and software-development kits (SDKs) support generalist programmers in using the full potential of heterogeneous hardware architectures consisting of a general-purpose CPU and specialized hardware like GPUs, machine-learning accelerators, or trusted enclaves. Fig. 3 presents this idea at a conceptual level.

Hybrid smart contracts are a first step along the way to this vision and to a concept we call *meta contracts*. Meta contracts are applications coded on a decentralized metalayer and implicitly encompass on-chain logic (smart contracts), as well as off-chain computation and connectivity among various blockchains and existing off-chain services. Given the need for language and compiler support, new security models, and conceptual and technical harmonization of disparate technologies, however, realization of a true decentralized metalayer is an ambitious goal to which we aspire over a long time horizon. It is nonetheless a helpful ideal model to keep in mind while reading this paper, not detailed here, but something we plan to focus on in our future work on Chainlink.

Scaling: A goal of preeminent importance in our evolving designs is enabling the Chainlink network to meet the growing scaling needs of the blockchain ecosystem.

With network congestion becoming a recurring problem in existing permissionless blockchains [86], new and more performant blockchain designs are coming into use, e.g., [103, 120, 203], as well as complementary layer-2 scaling technologies, e.g., [5, 12, 121, 141, 169, 186, 187]. Oracle services must achieve latencies and throughputs that meet the performance demands of these systems while minimizing on-chain fees (e.g., gas costs) for contract operators and ordinary users alike. With DONs, Chainlink functionality aims to go further and deliver performance high enough for purely web-based systems.

DONs derive much of their performance gain from their use of fast, committee-based or permissionless consensus protocols, which they combine with the blockchains they support. We expect many DONs with different configurations to run in parallel; different DApps and users can navigate tradeoffs in underlying consensus choices according to their application requirements.

DONs may be viewed in effect as layer-2 technologies. We expect that among other services, DONs will support the *Transaction Execution Framework* (TEF), which facilitates efficient integration of DONs and thus oracles with other high-performance layer-2 systems—e.g., *rollups*, systems which bundle transactions off chain to achieve performance improvements. We introduce the TEF in Section 6.

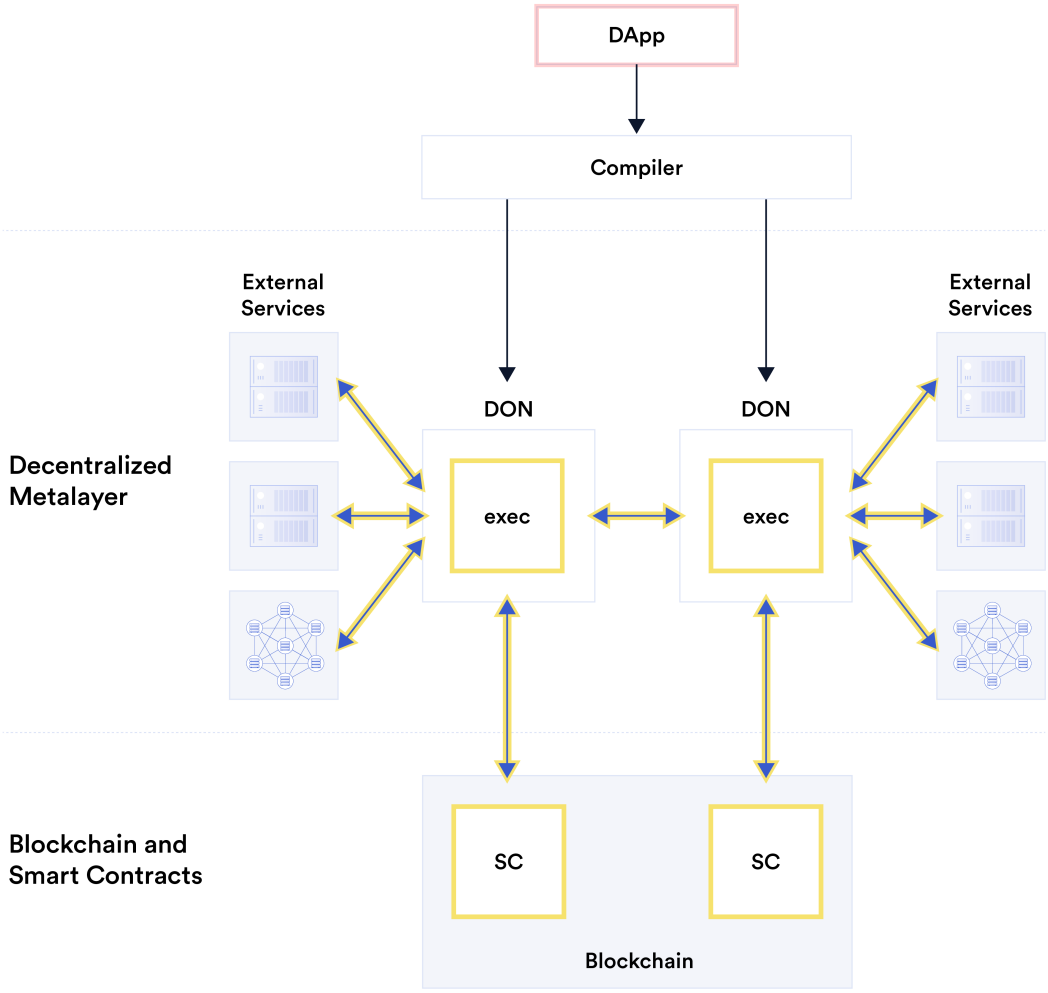

Figure 3: Conceptual figure showing ideal realization of a decentralized metalayer. For ease of development, a developer specifies a DApp, highlighted in pink, as a virtual application in a unified machine model. A decentralized-metalayer compiler automatically generates corresponding interoperating functionalities: smart contracts (denoted by SC), logic (denoted by exec) on DONs, adapters connecting to target external services, and so forth, as indicated in yellow highlight.

Fig. 4 shows conceptually how DONs improve blockchain (smart contract) scaling by concentrating transaction and oracle-report processing off chain, rather than on chain. This shift in the main locus of computation reduces transaction latency and fees while boosting transaction throughput.

Confidentiality: Blockchains provide unprecedented transparency for smart contracts and the applications they realize. But there is a basic tension between transparency and confidentiality. Today, for example, users' decentralized exchange trans-

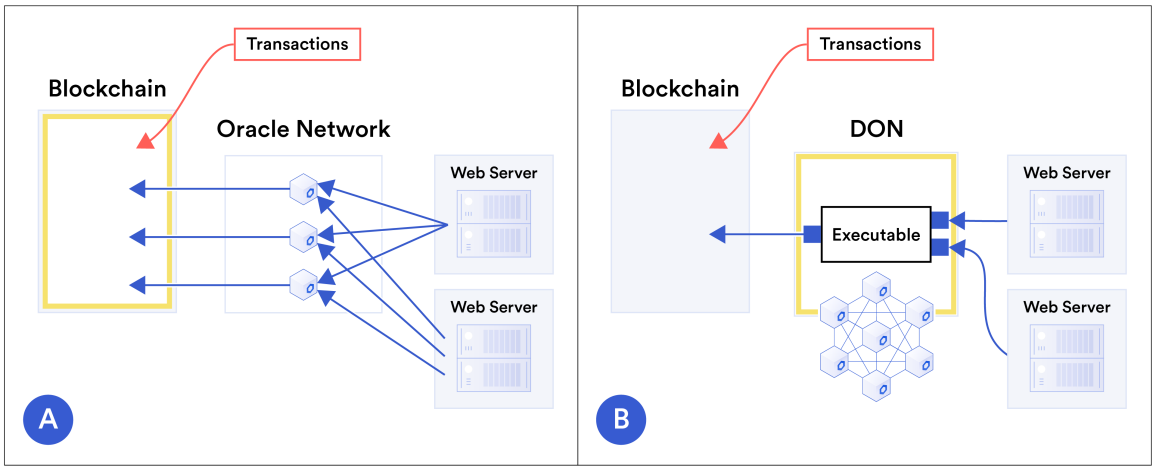

Figure 4: Conceptual figure showing how Decentralized Oracle Networks improve the scaling of blockchain-enabled smart contracts. Figure Ⓐ shows a conventional oracle architecture. Transactions are sent directly to the blockchain, as are oracle reports. Thus the blockchain, highlighted in yellow, is the main locus for transaction processing. Figure Ⓑ shows use of a DON to support contracts on the blockchain. A DON executable processes transactions along with data from external systems and forwards results—e.g., bundled transactions or contract state changes resulting from the transactions' effects—to the blockchain. The DON, highlighted in yellow, is thus the main locus for transaction processing.

actions are recorded on chain, making it easy to monitor exchange behavior, but also making users' financial transactions publicly visible. Similarly, data relayed to smart contracts remains on chain. This makes such data conveniently auditable, but acts as a disincentive for data providers wishing to furnish smart contracts with sensitive or proprietary data.

We believe that oracle networks will play a pivotal role in catalyzing next-generation systems that combine blockchains' innate transparency with new confidentiality protections. In this paper, we show how they will do so using three main approaches:

- *Confidentiality-preserving adapters:* Two technologies with planned deployment in Chainlink's networks, DECO [234] and Town Crier [233], enable oracle nodes to retrieve data from off-chain systems in ways that protect user privacy and data confidentiality. They will play a key role in the design of adapters for DONs. (See Section 3.6.2 for details on these two technologies.)

- *Confidential computation:* DONs can simply conceal their computation from relying blockchains. Using secure multi-party computation and/or trusted execution environments, stronger confidentiality is also possible in which DON nodes compute over data into which they themselves do not have visibility.

- *Support for confidential layer-2 systems:* The TEF is designed to support a variety of layer-2 systems, many of which use zero-knowledge proofs to provide various forms of transaction confidentiality.

We discuss these approaches in Section 3 (with additional details in Section 6, Appendix B.1, and Appendix B.2).

Fig. 5 presents a conceptual view of how sensitive data might flow from external sources to a smart contract by means of confidentiality-preserving adapters and confidential computation in a DON.

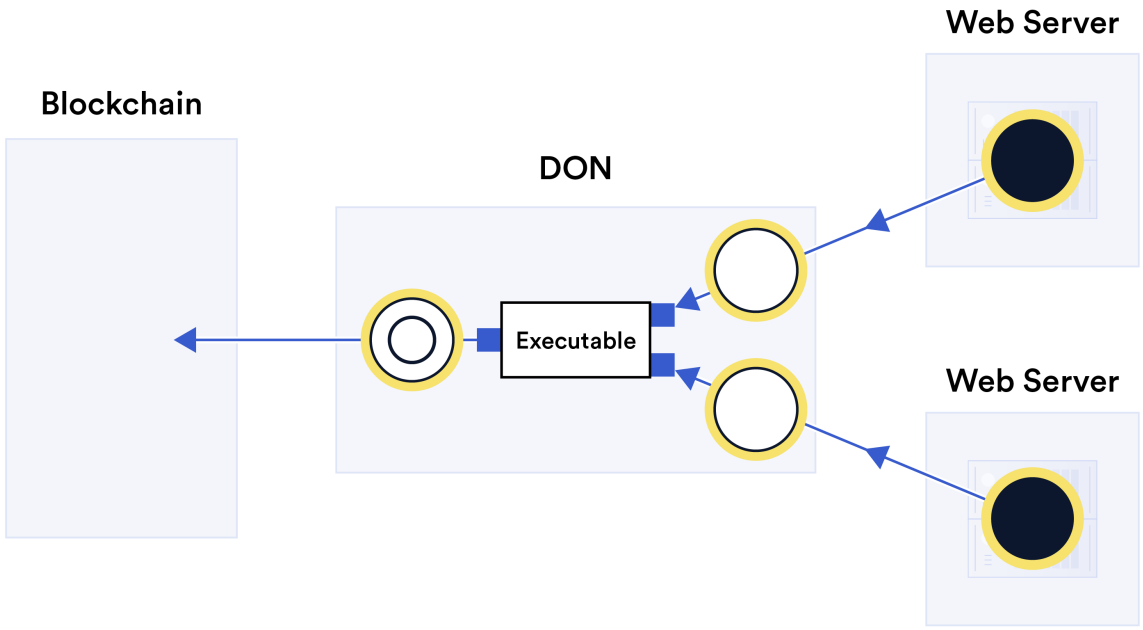

Figure 5: Conceptual diagram of confidentiality-preserving operations in a DON on sensitive data (highlighted in yellow). Sensitive source data (black circles) in web servers is extracted to the DON using confidentiality-preserving adapters (blue, double-arrowed lines). The DON receives *derived data* (hollow circles) from these adapters—the result of applying either a function or, e.g., secret-sharing, to the sensitive source data. An executable on the DON may apply confidential computation to derived data to construct a report (double circle), which it sends over an adapter to the blockchain.

We believe that powerful tools for handling confidential data will open up a whole range of applications. Among these are private decentralized (and centralized) finance, decentralized identity, credit-based on-chain lending, and more efficient and user-friendly know-your-customer and accreditation protocols, as we discuss in Section 4.

Order-fairness for transactions: Today's blockchain designs have a dirty little open secret: They are *ephemerally centralized*. Miners and validators can order trans-

actions however they choose. Transaction order can also be manipulated by users as a function of the network fees they pay (e.g., gas prices in Ethereum) and to some extent by taking advantage of fast network connections. Such manipulation can, for example, take the form of front-running, in which a strategic actor such as a miner observes a user's transaction and inserts its own exploitative transaction into an earlier position in the same block—effectively stealing money from the user by leveraging advance knowledge of the user's transaction. For example, a bot may place a buy order before a user's. It can then take advantage of the asset price increase induced by the user's trade.

Front-running by some bots that harms ordinary users—analogous to high-frequency trading on Wall Street—is already prevalent and well documented [90], as are related attacks such as back-running [159] and automated transaction mimicking [195]. Proposals to systematize order exploitation by miners have even surfaced recently [110].

Layer-2 technologies such as rollups don't solve the problem, but merely *re-centralize* ordering, placing it in the hands of the entity that creates a rollup.

One of our goals is to introduce into Chainlink a service called *Fair Sequencing Services* (FSS) [137]. FSS helps smart contract designers ensure fair ordering for their transactions and avoid front-running, back-running, and related attacks on user transactions as well as other types of transactions, such as oracle report transmission. FSS enables a DON to implement ideas such as the rigorous, temporal notion of *order-fairness* introduced in [144]. As an incidental benefit, FSS can also *lower users' network fees* (e.g., gas costs).

Briefly, in FSS, transactions pass through the DON, rather than propagating directly to a target smart contract. The DON orders the transactions and then forwards them to the contract.

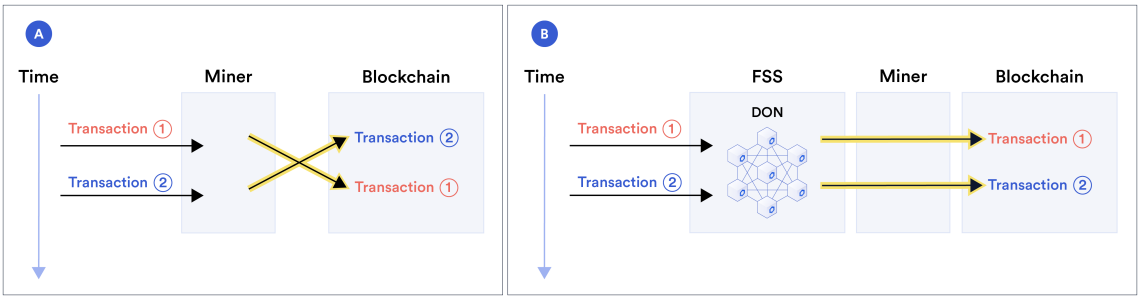

Figure 6: Example of how FSS is beneficial. Fig. Ⓐ shows how a miner, exploiting its centralized power to order transactions, may swap a pair of transactions: transaction ① arrives *before* ②, but the miner instead sequences it *after* ②. In contrast, Fig. Ⓑ shows how a DON *decentralizes* the ordering process among DON nodes. If a quorum of honest nodes receive ① *before* ②, the FSS causes ① to appear before ② on chain—preventing miner reordering by attaching contract-enforceable sequence numbers.

Fig. 6 compares standard mining with FSS. It shows how in standard mining,

the process of transaction ordering is *centralized* with the miner and thus subject to manipulation, such as reordering a pair of transactions with respect to their arrival times. In contrast, in FSS, the process is *decentralized* among DON nodes. Assuming a quorum of honest nodes, FSS helps enforce policies such as temporal ordering of transactions, reducing opportunities for manipulation by miners and other entities. Additionally, since users need not compete for preferential ordering based on gas price, they can pay relatively low gas prices (while transactions from the DON can be batched for gas savings).

Trust minimization: Our general aim in the design of DONs is to facilitate a highly trustworthy layer of support for smart contracts and other oracle-dependent systems by means of decentralization, cryptographic tools, and cryptoeconomic guarantees. A DON itself is decentralized, and users can choose from any available DON that supports the main chain on which they wish to operate or spawn additional DONs with committees of nodes they trust.

For some applications, however, particularly smart contracts, Chainlink users may favor a trust model that treats the main chain supported by a DON as more trustworthy than the DON itself. For such users, we already have or plan to incorporate into the architecture of the Chainlink network a number of mechanisms that enable contracts on a main chain to strengthen the security assurances provided by DONs, while at the same time also enforcing protections against the possibility of corrupted data sources such as the web servers from which the DON obtains data.

We describe these mechanisms in Section 7. They fall under five main headings:

- *Data-source authentication:* Tools that enable data providers to digitally sign their data and thereby strengthen the chain of custody between the origin and relying contract.
- *DON minority reports:* Flags issued by a minority subset of DON nodes that observes majority malfeasance in the DON.
- *Guard rails:* Logic on a main chain that detects anomalous conditions and pauses or halts contract execution (or invokes other remediations).
- *Trust-minimized governance:* Use of gradual-release updates to facilitate community inspection, as well as decentralized emergency interventions for rapid response to system failures.
- *Decentralized entity authentication:* Use of public-key infrastructure (PKI) to identify entities in the Chainlink network.

Fig. 7 presents a conceptual schematic of our trust-minimization goals.

Incentive-based (cryptoeconomic) security: Decentralization of report generation across oracle nodes helps ensure security even when some nodes are corrupted.

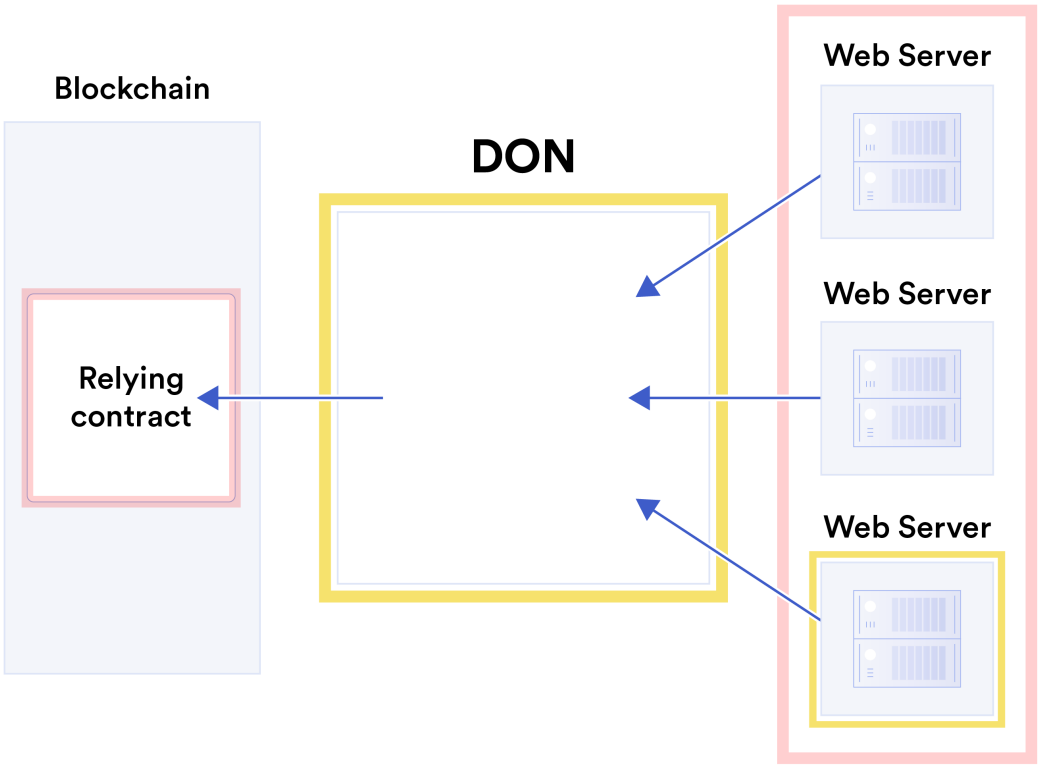

Figure 7: Conceptual depiction of Chainlink's trust-minimization goal, which is to minimize users' need for correct behavior of the DON and data sources such as web servers. Yellow highlights in the figure indicate trust-minimization loci: the DON and individual or minority sets of web servers. Pink highlights indicate system components that are highly trustworthy by assumption: contracts on the blockchain and a majority of web servers, i.e., web servers in the aggregate.

Equally important, though, is ensuring that nodes have a *financial incentive* to behave correctly. *Staking*, i.e., requiring nodes to provide deposits of LINK and slashing (confiscating) these deposits in case of misbehavior, will play a key role in Chainlink. It is an important incentive design already used in a number of blockchains, e.g., [81, 103, 120, 204].

Staking in Chainlink, however, looks very different from staking in standalone blockchains. Staking in blockchains aims to prevent attacks on consensus. It has a different goal in Chainlink: to ensure timely delivery of correct oracle reports. A well-designed staking system for an oracle network should render attacks such as bribery unprofitable for an adversary, even when the target is a smart contract with high monetary value.

In this paper, we present a general approach to staking in Chainlink with three key innovations:

1. A *powerful adversarial model* that encompasses attacks overlooked in existing approaches. One example is what we call *prospective* bribery. This is a form of bribery that determines which nodes receive bribes on a conditional basis, e.g., offers guaranteed bribes in advance to nodes that a staking mechanism selects at random for particular roles (such as triggering report adjudication).

2. *Super-linear staking impact*, meaning informally that to be successful, an adversary must have a budget $\$B$ *greater than the combined deposits of all oracle nodes*. More precisely, we mean that as a function of n, $\$B(n) \gg \$dn$ in a network of n oracle nodes each with a fixed deposit amount $\$d$ (more formally, $\$B(n)$ is asymptotically larger in n than $\$dn$). Fig. 8 gives a conceptual view of this property.

3. The *Implicit-Incentive Framework* (IIF), an incentive model we have devised to encompass empirically measurable incentives beyond explicit deposited staking funds, including nodes' *future fee opportunities*. The IIF extends the notion of stake beyond explicit node deposits.

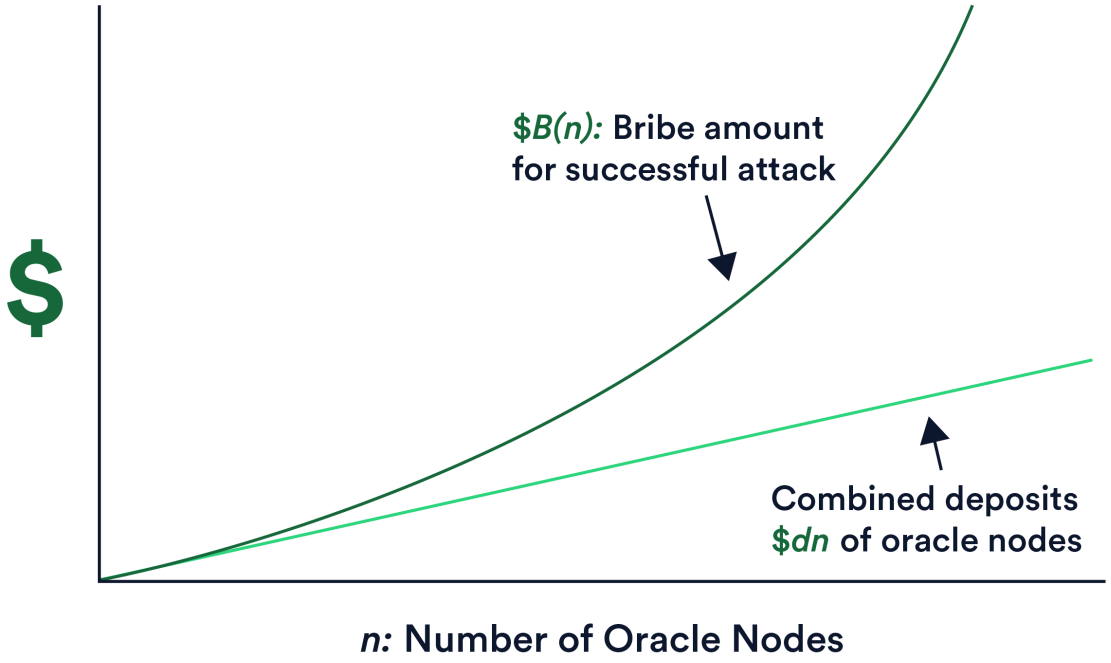

Figure 8: Conceptual diagram depicting super-linear scaling in Chainlink staking. The bribe $\$B(n)$ required by an adversary grows faster in n than the combined deposits $\$dn$ of all oracle nodes.

We show how the IIF and super-linear staking impact together induce what we call a *virtuous cycle of economic security* for oracle networks. When new users enter

the system, increasing potential future earnings from running Chainlink nodes, the marginal cost of economic security drops for current and future users. In a regime of elastic demand, this diminished cost incentivizes additional users to make use of the network, continuously perpetuating adoption in an ongoing virtuous cycle.

Note: While this whitepaper outlines important elements of our vision for the evolution of Chainlink, it is informal and includes few detailed technical specifics. We plan to release focused technical papers on additional features and approaches as they evolve. Furthermore, it is important to emphasize that many elements of the vision presented here (scaling improvements, confidentiality technologies, FSS, etc.) can and will be deployed in preliminary form even before advanced DONs become a basic feature of Chainlink.

1.3 Organization of this Paper

We present our security model and notation in Section 2 and outline the Decentralized Oracle Network API in Section 3. In Section 4, we present a number of examples of applications for which DONs provide an appealing deployment platform. Readers can learn most of the key concepts of the paper by reading up to this point.

The remainder of the paper contains further details. We describe Fair Sequencing Services (FSS) in Section 5 and the Transaction-Execution Framework (TEF) in Section 6. We describe our approach to trust minimization in Section 7. We consider some important DON deployment requirements, namely incremental rollout of features, dynamic ledger membership, and accountability in Section 8. Finally, in Section 9, we give an overview of our developing approach to incentive design. We conclude in Section 10.

To help readers who have limited familiarity with the concepts in this paper, we provide a glossary in Appendix A. We present further detail on the DON interface and functionality in Appendix B and present some example adapters in Appendix C. In Appendix D, we describe a cryptographic primitive for trust-minimized data-source authentication called functional signatures and introduce a new variant called *discretized functional signatures*. We discuss some considerations bearing on committee selection for DONs in Appendix F.

2 Security Model and Goals

A Decentralized Oracle Network is a distinct distributed system that we expect will initially be implemented typically—although not necessarily—by a committee-based consensus protocol and run by a set of oracle nodes. A DON is designed primarily to augment the capabilities of a smart contract on a main chain with oracle reports and other services, but it can provide those same supporting services to other non-blockchain systems, and thus need not be associated with a particular main chain.

The model and properties we consider are therefore largely independent of the use of the particular applications of a DON.

2.1 Current Architectural Model

It is important to emphasize that Chainlink today is not a monolithic service, but rather a permissionless framework within which it is possible to launch distinct, independent networks of oracle nodes [77]. Networks have heterogeneous sets of node operators and designs. They may also differ in terms of the types of services they provide, which can include, e.g., data feeds, Proof of Reserves, verifiable randomness, and so forth. Other differences can include the degree of decentralization, size of the network in terms of locked value it supports, and various service-level parameters, such as data frequency and accuracy.

Chainlink's permissionless model encourages the growth of an ecosystem in which providers specialize in the services they are best able to furnish to the community. This model is likely to result in lower costs to users and higher service quality than a model that requires all nodes and networks to provide a full range of services, an approach that can easily devolve into system-wide adoption of the services representing the least common denominator of resources available to nodes.

As Chainlink evolves toward DON-based designs in Chainlink 2.0, we continue to support the model of a permissionless, open framework, keeping in view the goal of providing users with a range of service choices that globally result in the best match with particular application requirements.

2.2 Consensus Assumptions

We use the term Decentralized Oracle Network to encompass the full functionality of the oracle system we describe: both the data structure that oracle nodes maintain and the core API layered on top of it.

We use the term *ledger* (lower case), denoted by $\mathcal{L}$, to mean the underlying data structure maintained by a DON and used to support the particular services it provides. We emphasize that our DON framework does not treat $\mathcal{L}$ as a freestanding system like a blockchain: Its purpose is to support blockchains and other systems. Blockchains are, of course, one way of realizing a trustworthy ledger, but there are others. We expect DONs in many cases to realize their underlying ledgers using Byzantine Fault Tolerant (BFT) systems, which considerably predate blockchains such as Bitcoin [174]. We use BFT-type notation and properties throughout the paper for convenience, although we emphasize that DONs can be realized using permissionless consensus protocols.

Conceptually, a ledger $\mathcal{L}$ is a bulletin board on which data is linearly ordered. We view a ledger generally as having a few key properties commonly ascribed to blockchains [115]. A ledger is:

- *Append-only:* Data, once added, cannot be removed or modified.

- *Public:* Anyone can read its contents, which are *consistent* across time in the view of all users.[4]

- *Available:* The ledger can always be written to by authorized writers and read by anyone in a timely way.

Alternative properties are possible in the ledger for a DON when realized by a committee. For instance, ledger write access might be restricted to certain users, as might read access for some applications, i.e., the ledger need not be public as defined above. Similarly, ledger rules might permit modification or redaction of data. We don't explicitly consider such variants in this paper, however.

The modular design of DONs can support any of a wide variety of modern BFT protocols, e.g., Hotstuff [231]. The exact choice will depend on trust assumptions and network characteristics among the oracle nodes. A DON could in principle alternatively use a highly performant permissionless blockchain for its ledger in its role supporting an equally scalable layer-2 or blockchain system. Similarly, hybridization is also possible: The DON could in principle be composed of nodes that are validators in an existing blockchain, e.g., in Proof-of-Stake systems in which committees are selected to execute transactions, e.g., [8, 81, 120, 146, 204]. This particular mode of operation requires that nodes operate in a dual-use manner, i.e., operate both as blockchain nodes and DON nodes. (See Section 8.2 for a discussion of techniques to ensure continuity in changing committees and Appendix F for some caveats on random committee selection.)

In practice, in modern BFT algorithms, nodes digitally sign messages on the ledger. We assume for convenience that $\mathcal{L}$ has an associated public key $\mathsf{pk}_{\mathcal{L}}$ and that its contents are signed by the corresponding private key. This general notation applies even when data on $\mathcal{L}$ are signed using threshold signatures.[5] Threshold signatures are convenient, as they enable a persistent identity for a DON even with changes of membership in the nodes running it. (See Appendix B.1.3.) We thus assume that $\mathsf{sk}_{\mathcal{L}}$ is secret-shared in a (k, n)-threshold manner for some security parameter k, e.g., $k = 2f + 1$ and $n = 3f + 1$, where f is the number of potentially faulty nodes. (By choosing k in this way, we ensure that faulty nodes can neither learn $\mathsf{sk}_{\mathcal{L}}$ nor mount a denial-of-service attack preventing its use.)

A message on $\mathcal{L}$ takes the form $M = (m, z)$, where m is a string and z a unique sequential index number. Where applicable, we write messages in the form $m = \langle \texttt{MessageType} : \text{payload} \rangle$. The message type `MessageType` is syntactic sugar that indicates the function of a particular message.

[4] In cases where a blockchain without finality realizes a ledger, inconsistency is typically abstracted away by disregarding insufficiently deep blocks or "pruning" [115].

[5] In practice, some code bases, e.g., LibraBFT [205], a variant of Hotstuff, have currently adopted multi-signatures, rather than threshold signatures, trading off reduced communication complexity for simpler engineering. With some added cost, oracle nodes can append threshold signatures to messages written to $\mathcal{L}$ even if the consensus protocol used for $\mathcal{L}$ doesn't employ them.

2.3 Notation

We denote the set of n oracle nodes running the ledger by $\mathcal{O} = \{\mathcal{O}_i\}_{i=1}^{n}$. Such a set of nodes is often called a *committee*. For simplicity, we assume that the set of oracles implementing DON functionality, i.e., services on top of $\mathcal{L}$, is identical with that maintaining $\mathcal{L}$, but they can be distinct. We let pk_i denote the public key of player $\mathcal{O}_i$, and sk_i the corresponding private key.

Most BFT algorithms require at least $n = 3f + 1$ nodes, where f is the number of potentially faulty nodes; remaining nodes are *honest*, in the sense that they follow the protocol exactly as specified. We refer to the committee $\mathcal{O}$ as *honest* if it meets this requirement, i.e., has greater than a 2/3-fraction of honest nodes. Unless otherwise stated, we assume that $\mathcal{O}$ is honest (and a static model of corruption). We use $\mathsf{pk}_{\mathcal{O}}$ / $\mathsf{sk}_{\mathcal{O}}$ interchangeably with $\mathsf{pk}_{\mathcal{L}}$ / $\mathsf{sk}_{\mathcal{L}}$, depending on the context.

We let $\sigma = \mathsf{Sig}_{\mathsf{pk}}[m]$ denote a signature on message m with respect to pk, i.e., using corresponding private key sk. Let $\mathsf{verify}(\mathsf{pk}, \sigma, m) \rightarrow \{\mathtt{false}, \mathtt{true}\}$ denote a corresponding signature verification algorithm. (We leave key generation implicit throughout the paper.)

We use the notation S to denote a data source and $\mathcal{S}$ to denote the full set of n_S sources in a given context. We denote by MAINCHAIN a smart-contract enabled blockchain supported by a DON. We use the term *relying contract* to denote any smart contract on MAINCHAIN that communicates with a DON, and use the notation SC to denote such a contract.

We generally assume that a DON supports a single main chain MAINCHAIN, although it can support multiple such chains, as we show in examples in Section 4. A DON can and typically will support multiple relying contracts on MAINCHAIN. (As noted above, a DON can alternatively support non-blockchain services.)

2.4 Note on Trust Models

As noted above, DONs may be built atop committee-based consensus protocols, and we expect they will commonly use such protocols. There are many strong arguments that one of the two alternatives, committee-based or permissionless blockchains, provides stronger security than the other.

It is important to recognize that the security of committee-based vs. permissionless decentralized systems is *incommensurable*. Compromising a PoW or a PoS blockchain via 51% attack requires that an adversary *obtain majority resources ephemerally and potentially anonymously*, for example by renting hash power in a PoW system. Such attacks in practice have already impacted several blockchains [200, 34]. In contrast, compromising a committee-based system means corrupting a threshold number (typically one-third) of its nodes, where the nodes may be publicly known, well resourced, and trustworthy entities.

On the other hand, committee-based systems (as well as "hybrid" permissionless systems that support committees) can support more functionality than strictly per-

missionless systems. This includes the ability to maintain persistent secrets, such as signing and/or encryption keys—one possibility in our designs.

We emphasize that DONs can in principle be built atop either a committee-based or permissionless consensus protocol and DON deployers may ultimately choose to adopt either approach.

Bolstering trust models: A key feature of Chainlink today is the ability of users to select nodes based on decentralized records of their performance histories, as discussed in Section 3.6.4. The staking mechanism and Implicit-Incentive Framework we introduce in Section 9 together constitute a broadly scoped and rigorous mechanism-design framework that will empower users with a greatly expanded ability to gauge the security of DONs. This same framework will also make it possible for DONs themselves to enforce various security requirements on participating nodes and ensure operation within strong trust models.

It is also possible using tools described in this paper for DONs to enforce special trust-model requirements, such as compliance with regulatory requirements. For example, using techniques discussed in Section 4.3, nodes can present evidence of node-operator characteristics, e.g., territory of operation, that can be used to help enforce compliance with, e.g., the General Data Protection Regulation (GDPR) Article 3 ("Territorial Scope") [105]. Such compliance can otherwise be challenging to meet in decentralized systems [45].

Additionally, in Section 7 we discuss plans to strengthen the robustness of DONs through trust-minimization mechanisms on the main chains they support.

3 Decentralized Oracle Network Interface and Capabilities

Here we briefly sketch the capabilities of DONs in terms of the simple but powerful interface they are designed to realize.

Applications on a DON are composed of *executables* and *adapters*. An executable is a program whose core logic is a deterministic program, analogous to a smart contract. An executable also has a number of accompanying *initiators*, programs that call entry points in the executable's logic when predetermined events occur—e.g., at certain times (like a cron job), when a price crosses a threshold, etc.—much like Keepers (see Section 3.6.3). *Adapters* provide interfaces to off-chain resources and may be called by either the initiators or core logic in executables. As their behavior may depend on that of external resources, initiators and adapters may behave non-deterministically.

We describe the DON developer interface and the functioning of executables and adapters in terms of the three resources typically used to characterize computing systems: networking, compute, and storage. We give a brief overview of each of these resources below and provide more details in Appendix B.

3.1 Networking

Adapters are interfaces through which executables running on a DON can send and receive data from off-DON systems. Adapters may be viewed as a generalization of the *adapters* used in Chainlink today [20]. Adapters may be bidirectional—i.e., they cannot just pull, but push data from a DON to a web server. They may also leverage distributed protocols as well as cryptographic functionality such as secure multi-party computation.

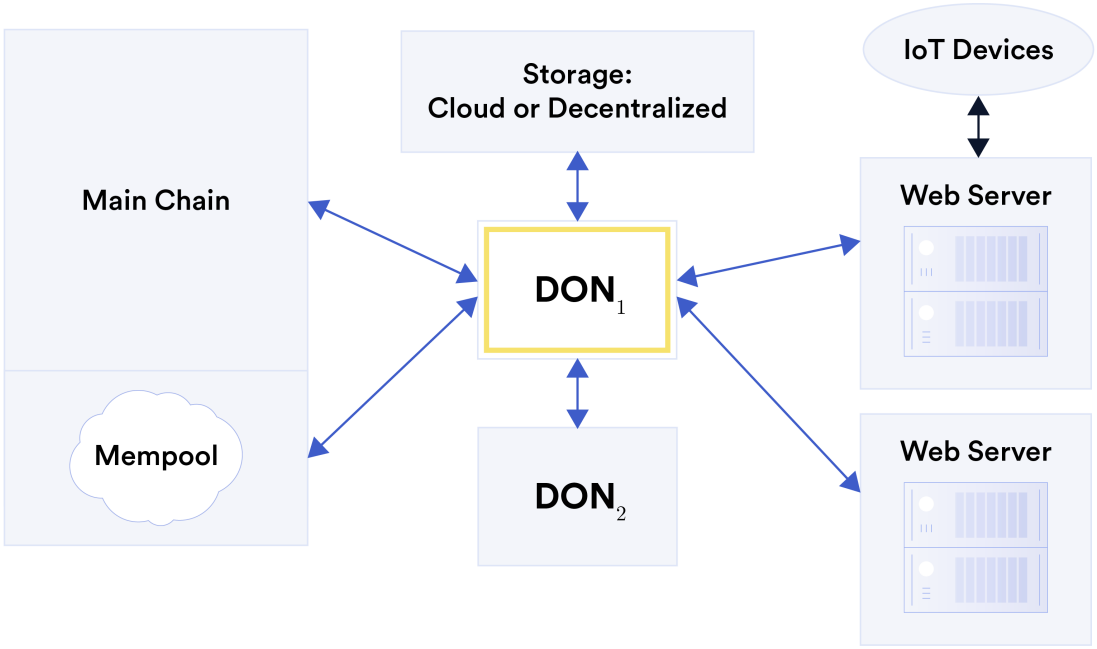

Figure 9: Adapters connecting a DON, denoted DON_1, with a range of different resources, including another DON, denoted DON_2, a blockchain (main chain) and its mempool, external storage, a web server, and IoT devices (via a web server).

Examples of external resources for which adapters might be created are shown in Fig. 9. They include:

- *Blockchains:* An adapter can define how to send transactions to a blockchain and how to read blocks, individual transactions, or other state from it. An adapter can also be defined for a blockchain's *mempool*. (See Section 3.5.)
- *Web servers:* Adapters can define APIs through which data may be retrieved from web servers, including legacy systems that are not specially adapted for interfacing with DONs. Such adapters can also include APIs to send data to such servers. The web servers to which a DON connects may serve as gateways to additional resources, such as Internet-of-Things (IoT) devices.

- *External storage:* An adapter can define methods to read and write to storage services outside the DON, such as a decentralized file system [40, 188] or cloud storage.

- *Other DONs:* Adapters can retrieve and transmit data between DONs.

We expect that initial deployments of DONs will include a set of *building block adapters* for such commonly used external resources and will further allow DON-specific adapters to be published by DON nodes. As smart contract developers write adapters today, we expect that they will build even more powerful adapters using this advanced functionality.

We expect that ultimately it will be possible for users to create new adapters in a permissionless manner.

Some adapters must be constructed in a way that ensures the *persistence* and *availability* of external resources controlled by a DON. For example, cloud storage may require maintenance of a cloud services account. Additionally, a DON can perform decentralized management of private keys on behalf of users (as in, e.g., [160]) and/or executables. Consequently, the DON is capable of controlling resources, such as cryptocurrency, that may be used, e.g., for sending transactions on a target blockchain.

See Appendix B.1 for further details on DON adapters, as Appendix C for a few example adapters.

3.2 Computation

An executable is the basic unit of code on a DON. An executable is a pair $\mathsf{exec} = (\mathsf{logic}, \mathsf{init})$. Here, logic is a deterministic program with a number of designated entry points $(\mathsf{logic}_1, \mathsf{logic}_2, \ldots, \mathsf{logic}_\ell)$ and init is a set of corresponding initiators $(\mathsf{init}_1, \mathsf{init}_2, \ldots, \mathsf{init}_e)$. To ensure the full auditability of the DON, an executable's logic uses the underlying ledger $\mathcal{L}$ for all inputs and outputs. Thus, for instance, any adapter data serving as input to an executable must be stored first on $\mathcal{L}$.

Initiators: Initiators in Chainlink today cause event-dependent job executions on Chainlink nodes [21]. Initiators in DONs function in much the same way. A DON initiator, however, is specifically associated with an executable. An initiator may depend on an *external event or state*, on the *current time*, or on a *predicate on DON state*. With their dependency on events, initiators may of course behave *non-deterministically* (as of course may adapters). An initiator can execute within individual DON nodes and so need not rely on an adapter. (See Example 1 below.)

Initiators are an important feature distinguishing executables from smart contracts. Because an executable can run in response to an initiator, it can effectively operate *autonomously*, as of course by extension can a hybrid contract incorporating the executable. One form of initiators today are Chainlink Keepers, which provide transaction

automation services, triggering smart contract execution—such as liquidation of undercollateralized loans and execution of limit-order trades—based on oracle reports.

Conveniently, initiators in DONs may also be viewed as a way of specifying the *service agreements* that apply to an executable, as they define the circumstances under which the DON must call it.

The following example illustrates how initiators work within an executable:

Example 1 (Deviation-triggered price feed)**.** *A smart contract* SC *may require fresh price-feed data (see Section 3.6.3) whenever there is a substantial change, e.g., 1%, in the exchange rate between a pair of assets, e.g., ETH-USD. Volatility-sensitive price feeds are supported in Chainlink today, but it is instructive to see how they can be realized on a DON by means of an executable* $\mathsf{exec}_{\mathsf{feed}}$.

The executable $\mathsf{exec}_{\mathsf{feed}}$ *maintains the most recent ETH-USD price* r *on* $\mathcal{L}$*, in the form of a sequence of* $\langle \texttt{NewPrice} : j, r \rangle$ *entries, where* j *is an index incremented with each price update.*

An initiator init_1 *causes each node* $\mathcal{O}_i$ *to monitor the current ETH-USD price for deviations of at least* 1% *from the most recently stored price* r *with index* j*. Upon detection of such a deviation,* $\mathcal{O}_i$ *writes its current view* r_i *of the new price to* $\mathcal{L}$ *using an entry of the form* $\langle \texttt{PriceView} : i, j+1, r_i \rangle$.

A second initiator init_2 *fires when at least* k *such* `PriceView`*-entries with new price values for index* $j+1$ *created by distinct nodes have accumulated on* $\mathcal{L}$*. Then,* init_2 *invokes an entry point* logic_2 *to compute the median* ρ *of the first* k *fresh, valid price-view values and writes a fresh value* $\langle \texttt{NewPrice} : j+1, \rho \rangle$ *to* $\mathcal{L}$ *. (Operationally, nodes may take turns as designated writers.)*

A third initiator init_3 *watches for* `NewPrice` *entries on* $\mathcal{L}$*. Whenever a new report* $\langle \texttt{NewPrice} : j, r \rangle$ *appears there, it invokes an entry point* logic_3 *that pushes* (j, r) *to* SC *using an adapter.*

As we have noted, an executable is similar in its capabilities to a smart contract. Apart from its higher performance, though, it differs from a typical main chain contract in two essential ways:

1. *Confidentiality:* An executable can perform *confidential* computation, i.e., a secret program may process cleartext inputs, or a published program may process secret input data, or a combination of both. In a simple model, secret data can be accessed by DON nodes, which conceal intermediate results and disclose only processed and sanitized values to MAINCHAIN. It is also possible to conceal sensitive data from DONs themselves: DONs are meant to support approaches such as multi-party computation, e.g., [42, 157], and trusted execution environments (TEEs) [84, 133, 152, 229] for this purpose.[6]

[6]By extension, keeping executables themselves secret with respect to DON nodes is also possible, although this is only practical today for non-trivial executables using TEEs.

2. *Supporting role:* An executable is meant to *support* smart contracts on a main chain, rather than replace them. An executable has several limitations that a smart contract does not:

   (a) *Trust model:* An executable operates within the trust model defined by the DON: Its correct execution relies on the honest behavior of $\mathcal{O}$. (A main chain can, however, provide some *guard rails* against DON malfeasance, as discussed in Section 7.3.)

   (b) *Asset access:* A DON can control an account on a blockchain—and thus control assets on it through an adapter. But a DON cannot authoritatively represent assets created on a main chain, e.g., Ether or ERC20 tokens, since their native chain maintains the authoritative record of their ownership.

   (c) *Lifecycle:* DONs may be stood up intentionally with limited lifetimes, as defined by on-chain service level agreements between DONs and the owners of relying contracts. Blockchains, in contrast, are meant to function as permanent archival systems.

See Appendix B.2 for further details on DON computation.

3.3 Storage

As a committee-based system, a DON can store moderate amounts of data persistently on $\mathcal{L}$ at much lower cost than a permissionless blockchain. Additionally, via adapters, DONs can reference external decentralized systems for data storage, e.g., Filecoin [85], and can thereby connect such systems to smart contracts. This option is particularly attractive for bulk data as a means of addressing the pervasive problem of "bloat" in blockchain systems.

DONs can thus store data locally or externally for use in their specifically supported services. A DON can additionally make use of such data in a *confidential* way, computing on data that is: (1) *secret-shared* across DON nodes or encrypted under a key managed by DON nodes in ways suitable for secure multi-party computation or partial or fully homomorphic encryption; or (2) *protected using a trusted execution environment*.

We expect that DONs will adopt a simple memory-management model common to smart-contract systems: An executable may only *write* to its own memory. Executables may, however, *read* from the memory of other executables.

See Appendix B.3 for further details on DON storage.

3.4 Transaction-Execution Framework (TEF)

DONs are intended to support contracts on a main chain MAINCHAIN (or on multiple main chains). The *Transaction-Execution Framework* (TEF), discussed in detail

in Section 6, is a general-purpose approach to the efficient execution of a contract SC across MAINCHAIN and a DON. The TEF is intended to support FSS and layer-2 technologies—simultaneously, if desired. Indeed, it is likely to serve as the main vehicle for use of FSS (and for that reason, we do not further discuss FSS in this section).

Briefly, in TEF an original target contract SC designed or developed for MAINCHAIN is refactored into a hybrid contract. This refactoring produces the two interoperating pieces of the hybrid contract: a MAINCHAIN contract SC_a that we refer to for clarity in the context of TEFs as an *anchor contract* and an executable exec_s on a DON. The contract SC_a custodies users' assets, executes authoritative state transitions, and also provides *guard rails* (see Section 7.3) against failures in the DON. The executable exec_s sequences transactions and provides associated oracle data for them. It can bundle transactions for SC_a in any of a number of ways—e.g., using validity-proof-based or optimistic rollups, confidential execution by the DON, etc.

We expect to develop tools that make it easy for developers to partition a contract SC written in a high-level language into pieces of MAINCHAIN and DON logic, SC_a and exec_s respectively, that compose securely and efficiently.

Using TEF to integrate high-performance transaction schemes with high-performance oracles is integral to our oracle scaling approach.

3.5 Mempool Services

An important application-layer feature that we intend to deploy on DONs in support of FSS and the TEF are *Mempool Services* (MS). MS may be viewed as an adapter, but one with first-class support.

MS provides support for legacy-compatible transaction processing. In this use, MS ingests from a main chain's mempool those transactions intended for a target contract SC on MAINCHAIN. MS then passes these transactions to an executable on the DON, where they are processed in the desired way. MS data can be used by the DON to compose transactions that can then be passed directly to SC from the DON or to another contract that calls SC. For example, the DON can forward transactions harvested via MS, or it can use MS data to set gas prices for transactions it sends to MAINCHAIN.

Because it monitors the mempool, MS can obtain transactions from users interacting directly with SC. Thus users may continue to generate their transactions using legacy software, i.e., applications unaware of the existence of MS and MS-configured contracts. (In this case, SC must be changed to ignore the original transactions and accept only those processed by the MS, so as to avoid double-processing.)

For use with a target contract SC, MS can be used with FSS and/or the TEF.

3.6 Stepping Stones: Existing Chainlink Capabilities

3.6.1 Off-Chain Reporting (OCR)

Off-Chain Reporting (OCR) [60] is a mechanism in Chainlink for oracle report aggregation and transmission to a relying contract SC. Recently deployed for Chainlink price feed networks, it represents a first step along the path to full DONs.

At its core, OCR is a BFT protocol designed to operate in a partially synchronous network. It ensures liveness and correctness in the presence of $f < n/3$ arbitrarily faulty nodes, guaranteeing the properties of Byzantine reliable broadcast, but it is not a complete BFT consensus protocol. Nodes do not maintain message logs that are consistent in the sense of representing a ledger that is identical in all of their views, and the leader of the protocol may equivocate without violating safety.

OCR is currently designed for a particular message type: medianized aggregation of (at least $2f+1$) values reported by participating nodes. It provides a key assurance on the reports it outputs for SC, called *attested* reports: The median value in an attested report is equal to or lies between values reported by two honest nodes. This property is the key *safety* condition for OCR. The leader may have some influence on the median value in an attested report, but only subject to this correctness condition. OCR can be extended to message types that aggregate values in different ways.

While the Chainlink network's liveness and correctness goals today do not require OCR to be a full-blown consensus protocol, they *do* require OCR to provide some additional forms of functionality not present in conventional BFT protocols, most notably:

1. *All-or-nothing off-chain report broadcast:* OCR ensures that an attested report is made quickly available to all honest nodes or none of them. This is a *fairness* property that helps ensure that honest nodes have an opportunity to participate in attested report transmission.

2. *Reliable transmission:* OCR ensures, even in the presence of faulty or malicious nodes, that all OCR reports and messages are transmitted to SC within a certain, pre-defined interval of time. This is a *liveness* property.

3. *Contract-based trust minimization:* SC filters out potentially erroneous OCR-generated reports, e.g., if their reported values deviate significantly from other recently received ones. This is a form of extra-protocol *correctness* enforcement.

All three of these properties will play a natural role in DONs. All-or-nothing off-chain (DON) broadcast is an important building block for cryptoeconomic assurances around reliable transmission, which is in turn an essential adapter property. Trust minimization in SC is a type of *guard rail*, as discussed in Section 7.3.

OCR also provides a basis for operational deployment and refinement of BFT protocols in Chainlink's oracle networks and thus, as noted above, a path to the full functionality of DONs.

3.6.2 DECO and Town Crier

DECO [234] and Town Crier [233] are a pair of related technologies currently being developed in Chainlink networks.

Most web servers today allow users to connect over a secure channel using a protocol called Transport Layer Security (TLS) [94]. (HTTPS indicates a variant of HTTP that is enabled with TLS, i.e., URLs prefixed with "https" denote use of TLS for security.) Most TLS-enabled servers have a notable limitation, though: They *don't digitally sign data*. Consequently, a user or *Prover* cannot present the data she receives from a server to a third party or *Verifier*, such as an oracle or smart contract, in a way that ensures the data's authenticity.

Even if a server were to digitally sign data, there remains a problem of confidentiality. A Prover may wish to redact or modify sensitive data before presenting it to a Verifier. Digital signatures are designed specifically to invalidate modified data, however. They thus prevent a Prover from making confidentiality-preserving alterations to data. (See Section 7.1 for more discussion.)

DECO and Town Crier are designed to allow a Prover to obtain data from a web server and present it to a Verifier in a way that ensures *integrity* and *confidentiality*. The two systems preserve integrity in the sense that they ensure that data presented by the Prover to the Verifier originates authentically from the target server. They support confidentiality in the sense of allowing the Prover to redact or modify data (while still preserving integrity).

A key feature of both systems is that they *do not require any modifications to a target web server*. They can operate with any existing TLS-enabled server. In fact, they are transparent to the server: From the viewpoint of the server, the Prover is establishing an ordinary connection.

The two systems have similar goals, but differ in their trust models and implementations as we now briefly explain.

DECO makes fundamental use of cryptographic protocols to achieve its integrity and confidentiality properties. While establishing a session with a target server using DECO, the *Prover* engages at the same time in an interactive protocol with the *Verifier*. This protocol enables the Prover to prove to the Verifier that it has received a given piece of data D from the server during its current session. The Prover can alternatively present the Verifier with a zero-knowledge proof of some property of D and thus not reveal D directly.

In a typical use of DECO, a user or a single node can export data D from a private session with a web server to all of the nodes in a DON. As a result, the full DON can attest to the authenticity of D (or a fact derived from D via a zero-knowledge proof). In addition to the example applications given later in the paper, this capability can be used to amplify high-integrity access to a data source by a DON. Even if only one node has direct access to a data source—due, for instance, to an exclusive arrangement with a data provider—it remains possible for the entire DON to attest to the correctness of

reports emitted by that node.

Town Crier relies on the use of a trusted execution environment (TEE) such as Intel SGX. Briefly, a TEE functions as a kind of black box that executes applications in a tamperproof and confidential way. In principle, even the owner of the host on which the TEE is running can neither (undetectably) alter a TEE-protected application nor view the application's state, which may include secret data.

Town Crier can achieve all of the functionality of DECO and more. DECO constrains the Prover to interaction with a single Verifier. In contrast, Town Crier enables a Prover to generate a *publicly verifiable* proof on data D fetched from a target server, i.e., a proof that anyone, even a smart contract, can verify directly. Town Crier can also securely ingest and make use of secrets (e.g., user credentials).

The main limitation of Town Crier is its reliance on TEEs. Production TEEs have recently been shown to have a number of serious vulnerabilities, although the technology is in its infancy and will undoubtedly mature. See Appendices B.2.1 and B.2.2 for further discussion of TEEs.

For a few example applications of DECO and Town Crier, see Sections 4.3, 4.5 and 9.4.3 and Appendix C.1.

3.6.3 Existing On-Chain Chainlink Services

Chainlink oracle networks provide a number of main services across a multiplicity of blockchains and other decentralized systems today. Further evolution as described in this whitepaper will endow these existing services with additional capabilities and reach. Three examples are:

Data feeds: Today, the majority of Chainlink users relying on smart contracts make use of *data feeds*. These are reports on the current value of key pieces of data according to authoritative off-chain sources. For example, *price feeds* are feeds reporting the prices of assets—cryptocurrencies, commodities, forex, indexes, equities, etc.—according to exchanges or data-aggregation services. Such feeds today already help secure billions of dollars in on-chain value through their use in DeFi systems such as Aave [147] and Synthetix [208]. Other examples of Chainlink data feeds include weather data for parametric crop insurance [75] and election data [93], among a number of others.

The deployment of DONs and other technologies described in this paper will enhance provision of data feeds in Chainlink networks in many ways, including:

- *Scaling:* OCR and subsequently DONs aim to enable Chainlink services to scale dramatically across the many blockchains they support. For example, we expect that DONs will help increase the number of data feeds provided by nodes using Chainlink from 100s to 1000s and beyond. Such scaling will help the Chainlink ecosystem achieve its goal of furnishing data relevant to smart contracts comprehensively and both meeting and anticipating existing and future needs.

- *Enhanced security:* By storing intermediate reports, DONs will retain records of node behaviors for high-fidelity monitoring and measurement of their performance and accuracy, enabling strong empirical grounding of reputation systems for Chainlink nodes. FSS and the TEF will enable price feeds to be incorporated with transaction data in flexible ways that prevent attacks such as front-running. (Explicit) staking will bolster existing cryptoeconomic protection of the security of data feeds.

- *Feed agility:* As blockchain-agnostic systems (indeed, more broadly, consumer-agnostic systems), DONs can facilitate the provision of data feeds to a multiplicity of relying systems. A single DON can push a given feed simultaneously to a set of different blockchains, eliminating the need for per-chain oracle networks and enabling rapid deployment of existing feeds on new blockchains and of additional feeds across currently serviced blockchains.

- *Confidentiality:* The ability to perform generalized computation in a DON enables computations on sensitive data to take place off chain, avoiding on-chain exposure. Additionally, using DECO or Town Crier, it is possible to achieve even stronger confidentiality, allowing report generation based on data that *isn't exposed even to DON nodes*. See Section 4.3 and Section 4.5 for examples.

Verifiable Random Functions (VRFs): Several types of DApps require a verifiably correct source of randomness to enable verification of their own fair operation. Non-Fungible Tokens (NFTs) are an example. The rarity of NFT features in Aavegotchi [23] and Axie Infinity [35] is determined by Chainlink VRF, as is the distribution of NFTs by means of ticket-based drawings in Ether Cards [102]; the wide variety of gaming DApps whose outcomes are randomized; and unconventional financial instruments, e.g., no-loss savings games such as PoolTogether [89], which allocate funds to random winners. Other blockchain and non-blockchain applications also require secure sources of randomness, including selection of decentralized-system committees and the execution of lotteries.

While block hashes can serve as a source of unpredictable randomness, they are vulnerable to manipulation by adversarial miners (and to some extent by users submitting transactions). Chainlink VRF [78] offers a considerably more secure alternative. An oracle has an associated private / public key pair $(\mathsf{sk}, \mathsf{pk})$ whose private key is maintained off chain and whose public key pk is published. To output a random value, it applies sk to an unpredictable seed x furnished by a relying contract (e.g., a block hash and DApp-specific parameters) using a function F, yielding $y = F_{\mathsf{sk}}(x)$ along with a proof of correctness. (See [180] for the VRF available on Chainlink.) What makes a VRF verifiable is the fact that with knowledge of pk, it is possible to check the correctness of the proof and therefore of y. The value y is consequently unpredictable to an adversary that cannot predict x or learn sk and infeasible for the service to manipulate.

Chainlink VRF may be viewed as just one of a family of applications that involve *custodianship of private keys off chain*. More generally, DONs can offer secure, decentralized storage of individual keys for applications and/or users, and combine this capability with generalized computation. The result is a host of applications, of which we give some examples in this paper, including key management for Proof of Reserves (see Section 4.1) and for users' decentralized credentials (and other digital assets) (see Section 4.3).

Keepers: Chainlink Keepers [87] enable developers to write code for decentralized execution of off-chain jobs, generally to trigger execution of relying smart contracts. Before the advent of Keepers, it was common for developers to operate such off-chain logic themselves, creating centralized points of failure (as well as considerable duplicated development effort). Keepers instead provide an easy-to-use framework for decentralized outsourcing of these operations, enabling shorter development cycles and strong assurance of liveness and other security properties. Keepers can support any of a wide variety of triggering goals, including price-dependent liquidation of loans or execution of financial transactions, time-dependent initiation of airdrops or payments in systems with yield harvesting, and so forth.

In the DON framework, initiators may be viewed as a generalization of Keepers in several senses. Initiators may make use of adapters, and thus can leverage a modularized library of interfaces to on-chain and off-chain systems, permitting rapid development of secure, sophisticated functionality. Initiators initiate computation in executables, which themselves offer the full versatility of DONs, permitting the wide range of decentralized services we present in this paper for on-chain and off-chain applications.

3.6.4 Node Reputation / Performance History

The existing Chainlink ecosystem natively documents the performance histories of contributing nodes on chain. This feature has given rise to a collection of reputation-oriented resources that ingest, filter, and visualize performance data on individual node operators and data feeds. Users can reference these resources to make informed decisions in their selection of nodes and to monitor the operation of existing networks. Similar capabilities will help users choose DONs.

For example, permissionless marketplaces today such as `market.link` allow node operators to list their oracle services and attest to their off-chain identities through services such as Keybase [4], which bind the profile of a node in Chainlink to its owner's existing domain names and social media accounts. Additionally, performance analytics tools, such as those available at `market.link` and `reputation.link`, allow users to view statistics on the historical performance of individual nodes, including their average response latency, the deviation of values in their reports from consensus values relayed on chain, revenue generated, jobs fulfilled, and more. These analytics tools also allow users to track the adoption of various oracle networks by other users, a form of

implicit endorsement of the nodes securing such networks. The result is a flat "web of trust" in which, by using particular nodes, high-value decentralized applications create a signal of their trust in those nodes that other users can observe and factor into their own node-selection decisions.

With DONs (and initially with OCR) comes a shift in transaction processing and contract activity more generally off chain. A decentralized model for recording node performance remains possible within the DON itself. Indeed, the high performance and data capacity of DONs make it possible to construct records in a fine-grained way and also to perform decentralized computation on these records, yielding trustworthy summaries that can be consumed by reputation services and checkpointed on MAINCHAIN.

While it is possible for a DON in principle to misrepresent the behavior of constituent nodes if a large fraction of nodes is corrupted, we note that the *collective performance of a DON itself* in delivering on-chain data is visible on MAINCHAIN and thus cannot be misrepresented. Additionally, we plan to explore mechanisms that incentivize accurate internal reporting of node behaviors in a DON. For example, by reporting the subset of high-performing nodes that most quickly return data contributing to a report relayed on chain, a DON creates an incentive for nodes to contest incorrect reports: Incorrectly including nodes in this subset means incorrectly *excluding* nodes that should have been included and therefore invalidly penalizing them. Repeated reporting failures by a DON would also create an incentive for honest nodes to leave the DON.

Decentralized compilation of accurate performance histories and the consequent ability of users to identify high-performing nodes and for node operators to build reputations are important distinguishing features of the Chainlink ecosystem. We show in Section 9 how we can reason about them as a key piece of a rigorous and expansive view of the economic security provided by DONs.

4 Decentralized Services Enabled by Decentralized Oracle Networks

To illustrate the versatility of DONs and how they enable a host of new services, we present five examples of DON-based applications in this section and describe the hybrid contracts that realize them: (1) *Proof of Reserves*, a form of cross-chain service; (2) *Interfacing with enterprise / legacy systems*, that is, creating a middleware-based abstraction layer that facilitates development of blockchain applications with minimal blockchain-specific code or expertise; (3) *Decentralized identity*, tools enabling users to obtain and manage their own identity documents and credentials; (4) *Priority channels*, a service that ensures timely inclusion of critical-infrastructure transactions (e.g., oracle reports) on a blockchain; and (5) *Confidentiality-preserving DeFi*, that is, financial smart contracts that conceal the sensitive data of participating parties. Here, we

use SC to denote the $\mathsf{MAINCHAIN}$ part of a hybrid contract and describe the DON component separately or in terms of an executable exec.

4.1 Proof of Reserves

For many applications, it is useful to relay state between or among blockchains. A popular application of such services is cryptocurrency *wrapping*. Wrapped coins such as WBTC [15] are becoming a popular asset in Decentralized Finance (DeFi). They involve depositing the "wrapped" backing asset on its source blockchain $\mathsf{MAINCHAIN}^{(1)}$ and creating a corresponding token on a different, target blockchain $\mathsf{MAINCHAIN}^{(2)}$. For example, WBTC is an ERC20 token on the Ethereum blockchain that corresponds to BTC on the Bitcoin blockchain.

Because contracts on $\mathsf{MAINCHAIN}^{(2)}$ do not have direct visibility into $\mathsf{MAINCHAIN}^{(1)}$, they must rely explicitly or implicitly on an oracle to report on deposits of the wrapped asset in a smart contract, producing what is sometimes called a *Proof of Reserves*. In WBTC [15], for example, custodian BitGo holds BTC and issues WBTC, with the Chainlink network providing Proofs of Reserve [76].

A DON can itself provide a Proof of Reserves. With a DON, however, it is possible to go further. A DON can manage secrets and, through use of appropriate adapters, can transact on any desired blockchain. Consequently, it is possible for the DON to act as one among a number of custodians—or even as a sole, decentralized custodian—for a wrapped asset. DONs can thereby serve as a platform to enhance the security of existing services that use Proofs of Reserves.

For example, suppose that $\mathsf{MAINCHAIN}^{(1)}$ is Bitcoin and $\mathsf{MAINCHAIN}^{(2)}$ is Ethereum. On $\mathsf{MAINCHAIN}^{(2)}$, a contract SC issues tokens representing wrapped BTC. The DON controls a BTC address $\mathsf{addr}^{(1)}_{\mathsf{DON}}$. To wrap BTC, then, a user $\mathcal{U}$ sends X BTC from $\mathsf{addr}^{(1)}_{\mathcal{U}}$ to $\mathsf{addr}^{(1)}_{\mathsf{DON}}$ along with a $\mathsf{MAINCHAIN}^{(2)}$-address $\mathsf{addr}^{(2)}_{\mathcal{U}}$. The DON monitors $\mathsf{addr}^{(1)}_{\mathsf{DON}}$ via an adapter to $\mathsf{MAINCHAIN}^{(1)}$. On observing $\mathcal{U}$'s deposit, with sufficiently high-probability confirmation, it sends a message to SC via an adapter to $\mathsf{MAINCHAIN}^{(2)}$. This message instructs SC to mint X tokens for $\mathsf{addr}^{(2)}_{\mathcal{U}}$.

For $\mathcal{U}$ to release X tokens, the reverse happens. On $\mathsf{MAINCHAIN}^{(1)}$, however, $\mathsf{addr}^{(1)}_{\mathsf{DON}}$ sends X BTC to $\mathsf{addr}^{(1)}_{\mathcal{U}}$ (or to another address, if thus requested by the user). These protocols can be adapted, of course, to work with exchanges, rather than directly with users.

4.2 Interfacing with Enterprise / Legacy Systems

DONs can serve as bridges between and among blockchains, as in the example of Proof of Reserves, but another objective is for them to act as bidirectional bridges between blockchains and legacy systems [176] or blockchain-like systems such as central bank digital currencies [30].

Enterprises face a number of challenges in connecting their existing systems and processes to decentralized systems, including:

- *Blockchain agility:* Blockchain systems change rapidly. An enterprise may confront the rapid new appearance or rise in popularity of blockchains on which counterparties wish to conduct transactions, but for which the enterprise has no support in its existing infrastructure. In general, blockchains' dynamism makes it difficult for individual enterprises to remain abreast of the full ecosystem.

- *Blockchain-specific development resources:* For many organizations, hiring or incubating cutting-edge blockchain expertise is difficult, particularly in view of the challenge of agility.

- *Private-key management:* Managing private keys for blockchains or cryptocurrencies requires operational expertise distinct from that of traditional cybersecurity practices and unavailable to many enterprises.

- *Confidentiality:* Enterprises are leery of exposing their sensitive and proprietary data on chain.

To address the first three of these difficulties, developers can simply use a DON as a secure middleware layer to enable enterprise systems to read from or write to blockchains. The DON can abstract away detailed technical considerations such as gas dynamics, chain reorganization, and so forth, for both developers and users. By presenting a streamlined blockchain interface to enterprise systems, a DON can thus considerably simplify the development of blockchain-aware enterprise applications, removing the burden from enterprises of acquiring or incubating blockchain-specific development resources.

Such use of DONs is especially attractive in that it enables enterprise developers to create smart-contract applications that are largely *blockchain agnostic*. As a result, the larger the set of blockchains for which a DON is instrumented to act as middleware, the larger the set of blockchains to which enterprise users can gain easy access. Developers can port applications from existing blockchains to new ones with minimal modification to their internally developed applications.

To address the additional problem of confidentiality, developers can appeal to the tools we introduce in this paper and expect to deploy in support of DON applications. These include DECO and Town Crier Section 3.6.2 as well as confidentiality-preserving API modifications discussed in Section 7.1.2 and a number of application-specific approaches covered in the remainder of this section. These DON systems can provide high-integrity, on-chain attestations about enterprise system state without revealing sensitive enterprise source data on chain.

4.3 Decentralized Identity

Decentralized identity is a general term for the notion that users should be able to obtain and manage their own credentials, rather than relying on third parties to do so. Decentralized credentials are attestations to attributes or assertions of the holder,

which are often called *claims*. Credentials are digitally signed by entities, often called *issuers*, that can authoritatively associate claims with users. In most proposed schemes, claims are associated with a Decentralized Identifier (DID), a universal identifier for a given user. Credentials are bound to a public key whose private key the user holds. The user can thus prove possession of a claim using her private key.

Visionary as decentralized identity is, existing and proposed schemes, e.g., [14, 92, 129, 216], have three severe limitations:

- *Lack of legacy compatibility:* Existing decentralized identity systems rely on a community of authorities, called *issuers*, to produce DID credentials. Because existing web services do not generally digitally sign data, issuers must be launched as special-purpose systems. Because there is no incentive to do this without a decentralized-identity ecosystem, a chicken-and-the-egg problem results. In other words, it's unclear how to bootstrap an issuer ecosystem.

- *Unworkable key management:* Decentralized identity systems require users to manage private keys, something that experience with cryptocurrency has shown to be an unworkable onus. It is estimated that some 4,000,000 Bitcoin have been lost forever because of key management failures [194], and many users store their crypto assets with exchanges [193], thereby undermining decentralization.

- *Lack of privacy-preserving Sybil resistance:* A basic security requirement of applications such as voting, fair allocation of tokens during token sales, etc. is that users be unable to assert multiple identities. Existing decentralized identity proposals require users to reveal their real-world identities in order to achieve such Sybil resistance, thereby undermining important privacy assurances.

It is possible to address these problems using a combination of a committee of nodes performing distributed computation within a DON and the use of tools such as DECO or Town Crier, as shown in a system called CanDID [160].

DECO or Town Crier can by design turn existing web services *without modification* into confidentiality-preserving credential issuers. They enable a DON to export relevant data for this purpose into a credential while concealing sensitive data that should not appear in the credential.

In addition, to facilitate key recovery for users, thus addressing the key-management problem, a DON can allow users to store private keys in secret-shared form. Users can recover their keys by proving to the nodes in the DON—similarly, using Town Crier or DECO—an ability to log into accounts with a set of predetermined web providers (e.g., Twitter, Google, Facebook). The benefit of using Town Crier or DECO, as opposed to OAUTH, is user privacy. Those two tools enable a user to avoid revealing to the DON a web provider identifier—from which real-world identities can often be derived.

Finally, to provide Sybil resistance, as shown in [160], it is possible for a DON to perform a privacy-preserving transformation of unique real-world identifiers for users (e.g., Social Security Numbers (SSNs)) into on-chain identifiers upon user registration.

The system can thereby detect duplicate registrations without sensitive data such as SSNs being revealed to individual DON nodes.[7]

A DON can provide any of these services on behalf of external decentralized identity systems on permissionless or permissioned blockchains, e.g., instances of Hyperledger Indy [129].

Example application: KYC: Decentralized identity holds promise as a means to streamline requirements for financial applications on blockchains while improving user privacy. Two challenges it can help address are accreditation and compliance obligations under anti-money-laundering / know-your-customer (AML / KYC) regulations.

AML regulations in many countries require financial institutions (and other businesses) to establish and verify the identities of individuals and businesses with which they perform transactions. KYC forms one component of a financial institution's broader AML policy, which also typically involves monitoring user behaviors and watching fund flows, among other things.

KYC typically involves user presentation of identity credentials in some form (e.g., entry into an online web form, holding up an identity document in front of a user's face in a video session, etc.). Secure creation of and presentation of decentralized credentials could in principle be a beneficial alternative in several respects, namely by: (1) Making the KYC process more efficient for users and financial institutions, because once a credential is obtained, it could be presented seamlessly to any financial institution; (2) Reducing fraud by reducing opportunities for identity theft through compromise of personally identifiable information (PII) and spoofing during video verification; and (3) Reducing the risk of PII compromise in financial institutions, as users retain control of their own data.

Given the multi-billion-dollar penalties paid by financial institutions for AML compliance failures, and the many financial institutions spending millions of dollars annually on KYC, improvements could yield considerable savings for financial institutions and, by extension, for consumers [196]. While the traditional financial sector is slow to adopt new compliance tools, DeFi systems are increasingly embracing it [43].

Example application: Under-collateralized loans: Most DeFi applications that support lending today originate only *fully collateralized* loans. These are loans made to borrowers who deposit cryptocurrency assets of value exceeding that of the loans. Interest has arisen recently in what the DeFi community generally refers to as *under-collateralized* loans. These, by contrast, are loans for which the corresponding collateral has value that is less than that of the principal of the loan. Under-collateralized loans resemble loans often made by traditional financial institutions. Rather than relying on deposited collateral as a guarantee of loan repayment, they instead base lending decisions on the credit histories of borrowers.

[7] This transformation relies on a distributed pseudorandom function (PRF).

Under-collateralized loans constitute a nascent but growing part of the DeFi lending market. They rely upon mechanisms like those employed by traditional financial institutions, such as legal contracts [91]. An essential requirement for their growth will be the ability to furnish data on user creditworthiness—a key factor in conventional lending decisions—to DeFi systems in a way that provides strong integrity, i.e., assurance of correct data.

A DON-enabled decentralized identity system would enable would-be borrowers to generate high-assurance credentials attesting to their creditworthiness while preserving the confidentiality of sensitive information. Specifically, borrowers can generate these credentials based on records from authoritative online sources while exposing only the data attested to by the DON, without exposing other, potentially sensitive data. For example, a borrower can generate a credential indicating that her credit score with a set of credit bureaus exceeds a particular threshold (e.g., 750), without revealing her precise score or any other data in her records. Additionally, if desired, such credentials can be generated *anonymously*, i.e., the user's name can be treated as sensitive data and itself not exposed to oracle nodes or in her decentralized credential. The credential itself can be used on chain or off chain, depending on the application.

In summary, a borrower can provide essential information to lenders on their credit histories with strong integrity and without risk of exposure of unnecessary, sensitive data.

A borrower can also provide a variety of other confidentiality-preserving credentials helpful in making lending decisions. For example, credentials can attest to a borrower's possession of (off-chain) assets, as we show in our next example.

Example application: Accreditation: Many jurisdictions limit the class of investor to which unregistered securities may be sold. For example, in the U.S., SEC Regulation D stipulates that to be accredited for such investment opportunities, an individual must possess a net worth of $1 million, meet certain minimum income requirements, or have certain professional qualifications [209, 210]. Current accreditation processes are cumbersome and inefficient, often requiring a letter of attestation from an accountant, or similar evidence.

A decentralized identity system would enable users to generate credentials from existing online financial services accounts that prove compliance with accreditation regulations, facilitating a more efficient and privacy-preserving KYC process. The privacy-preserving properties of DECO and Town Crier, moreover, would allow these credentials to be generated with a strong assurance of integrity without directly revealing details of a user's financial status. For example, a user could generate a credential proving that she has a net worth of at least $1 million without revealing any additional information about her financial status.

4.4 Priority Channels

Priority channels are a useful new service that is easy to build using a DON. Their

goal is to deliver select, high-priority transactions in a timely way on MAINCHAIN during periods of network congestion. Priority channels may be viewed as a form of futures contract on block space and thus as a *cryptocommodity*, a term coined as part of Project Chicago [61, 136].

Priority channels are intended specifically for miners to enable infrastructure services, such as oracles, governance functions for contracts, etc.—not for ordinary user-level activities such as financial transactions. In fact, as designed here, a priority channel implemented by less than 100% of the mining power in the network can only provide loose bounds on delivery times, preventing its use for highly speed-dependent goals such as front-running.

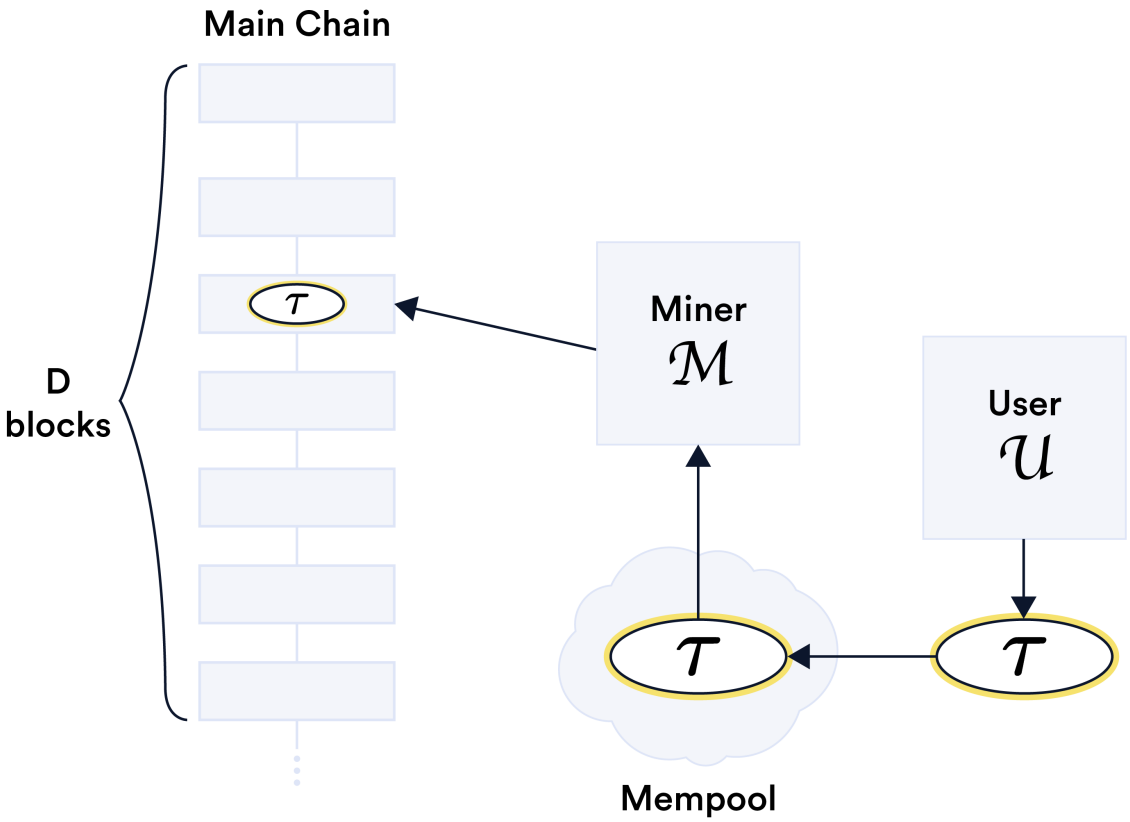

Figure 10: A priority channel is a guarantee by a miner $\mathcal{M}$—or, more generally, a set of miners $\mathcal{M}$—to a user $\mathcal{U}$ that her transaction τ will be mined within D blocks of inclusion in the mempool. A contract SC can use DON monitoring to enforce the service terms of the channel.

A priority channel takes the form of an agreement between a miner or set of miners (or mining pools) $\mathcal{M}$ that provides the channel and a user $\mathcal{U}$ that pays a fee for access. $\mathcal{M}$ agrees that when $\mathcal{U}$ submits a transaction τ to the mempool (with any gas price,

but a pre-agreed-upon gas limit), $\mathcal{M}$ will place it on chain within the next D blocks.[8] The idea is depicted schematically in Fig. 10.

Priority-channel contract description: A priority channel may be realized as a hybrid smart contract roughly as follows. We let SC denote the logic on MAINCHAIN and that on the DON by exec.

SC accepts a deposit / stake $\$d$ from $\mathcal{M}$ and an advance payment $\$p$ from $\mathcal{U}$. A DON executable exec monitors the mempool, triggering on placement of a transaction by $\mathcal{U}$. It sends a `success` message to SC if $\mathcal{U}$ submits a transaction that $\mathcal{M}$ mines in a timely way and a `failure` message in case of a service failure.

SC sends payment $\$p$ to $\mathcal{M}$ given a *success* message and sends all remaining funds, including $\$d$, to $\mathcal{U}$ if it receives a `failure` message. Upon successful termination, it releases deposit $\$d$ to $\mathcal{M}$.

The miner $\mathcal{M}$ can of course provide priority channels simultaneously to multiple users and can open a priority channel with $\mathcal{U}$ for a pre-agreed-upon number of messages.

4.5 Confidentiality-Preserving DeFi / Mixicles

Today, DeFi applications [1] provide little to no confidentiality for users: All transactions are visible on chain. Various zero-knowledge-based approaches, e.g., [149, 217], can provide transaction privacy, and the TEF is general enough to support them. But these approaches are not comprehensive, and do not, for example, typically conceal the asset on which a transaction is based.

The broad set of computational tools we ultimately intend to support in DONs will enable privacy in a number of different ways that can plug such gaps, helping complement the privacy assurances of other systems. For example, Mixicles, a confidentiality-preserving DeFi instrument proposed by Chainlink Labs researchers [135], can conceal the asset type backing a financial instrument, and fit very naturally into the DON framework.

Mixicles are most easily explained in terms of their use to realize a simple binary option. A binary option is a financial instrument in which two users, which we'll refer to here for consistency with [135] as *players*, bet on an event with two possible outcomes, e.g., whether or not an asset exceeds a target price at a predesignated time. The following example illustrates the idea.

Example 2. *Alice and Bob are parties to a binary option based on the value of an asset called Carol's Bubble Token (*CBT*). Alice bets that* CBT *will have a market price of at least 250 USD at time T = noon on 21 June 2025; Bob bets the reverse. Each player deposits 100 ETH by a prespecified deadline. The player with the winning position receives 200 ETH (i.e., gains 100 ETH).*

[8] D must of course be large enough to ensure that $\mathcal{M}$ can comply with high probability. For instance, if $\mathcal{M}$ controls 20% of the mining power in the network, it might choose $D = 100$, ensuring a failure probability of $\approx 2 \times 10^{-10}$, i.e., less than one in a billion.

Given an existing Chainlink oracle network $\mathcal{O}$, it is easy to implement a smart contract SC that realizes the agreement in Example 2. The two players each deposit 100 ETH in SC. Sometime after T, a query q is sent to $\mathcal{O}$ requesting the price r of CBT at time T. $\mathcal{O}$ sends a report r of this price to SC. SC then sends money to Alice if $r \geq 250$ and Bob if not. This approach, however, reveals r on chain—making it easy for an observer to deduce the asset underlying the binary option.

In the terminology of Mixicles, it is helpful to think conceptually of the outcome of SC in terms of a *Switch* that transmits a binary value computed as a predicate $\mathsf{switch}(r)$. In our example, $\mathsf{switch}(r) = 0$ if $r \geq 250$; given this outcome, Alice wins. Otherwise $\mathsf{switch}(r) = 1$ and Bob wins.

A DON can realize a basic Mixicle as a hybrid contract by running an executable exec that computes $\mathsf{switch}(r)$ and reports it on chain to SC. We show this construction in Fig. 11.

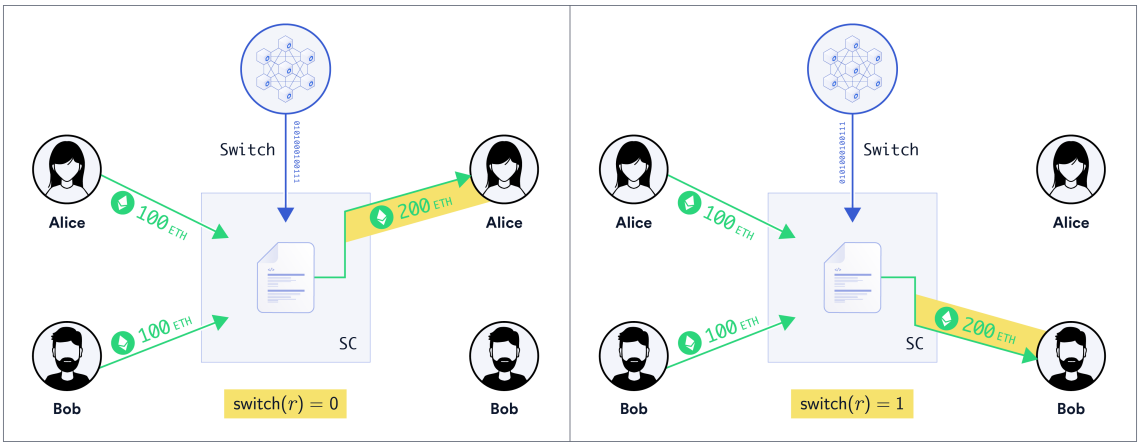

Figure 11: Diagram of basic Mixicle in Example 2. To provide on-chain secrecy for report r, and thus the asset underlying the binary option, the oracle sends to the contract SC via Switch only the binary value $\mathsf{switch}(r)$.

We specify an adapter ConfSwitch in Appendix C.3 that makes it easy to achieve this goal in a DON. The basic idea behind ConfSwitch is quite simple. Instead of reporting the value r, ConfSwitch reports only the binary switch value $\mathsf{switch}(r)$. SC can be designed to make a correct payment based on $\mathsf{switch}(r)$ alone, and $\mathsf{switch}(r)$ by itself reveals no information about the underlying asset—CBT in our example. Additionally, by placing a ciphertext on (q, r) on the ledger encrypted under $\mathsf{pk}_{\mathsf{aud}}$, the public key of an auditor, the adapter ConfSwitch creates a confidentiality-preserving audit trail.

The basic Mixicle we've chosen for simplicity to describe here conceals only the asset and bet behind the binary option in our example. A full-blown Mixicle [135] can provide two forms of confidentiality. It conceals from observers: (1) What event the players bet on (i.e., q and r) but *also* (2) Which player won the bet.

Since Mixicles are executed on MAINCHAIN, either one player would need to relay $\mathsf{switch}(r)$ from the DON to MAINCHAIN, or an executable exec could be created that

is triggered on output by ConfSwitch and calls another adapter to send switch(r) to MAINCHAIN.

A third, subtle type of confidentiality is also worth considering. In a basic implementation of ConfSwitch, $\mathcal{O}$ is running the adapter on the DON and thus learns the asset—CBT in our example—and thus the nature of the binary option. As discussed in Appendix C.3, however, it is additionally possible to use DECO or Town Crier to conceal even this information from $\mathcal{O}$. In this case, the $\mathcal{O}$ learns no more information than a public observer of SC.

For further details on Mixicles, we refer readers to [135].

5 Fair Sequencing Services

One important service that we expect DONs will offer that leverages their networking, computation, and storage capabilities is called *Fair Sequencing Services* (FSS). Although FSS may be viewed simply as an application realized within the DON framework, we highlight it as a service that we believe will be in high demand across blockchains, and which we expect the Chainlink network to support actively.

When executed on public blockchain networks, many of today's DeFi applications reveal information that can be exploited by users to their own benefit, analogous to the kind of insider leaks and manipulation opportunities that are pervasive in existing markets [64, 155]. FSS instead paves the way toward a *fair* DeFi ecosystem. FSS helps developers to build DeFi contracts that are protected from market manipulation resulting from information leakage. Given the problems we highlight below, FSS is especially attractive for layer-2 services and fits within the framework for such services that we discuss in Section 6.

The challenge: In existing permissionless systems, transactions are ordered entirely at the discretion of miners. In permissioned networks, the validator nodes may exert the same power. This is a form of largely unrecognized ephemeral *centralization* in otherwise decentralized systems. A miner can (temporarily) censor transactions for its own benefit [171] or reorder them to maximize its own gain, a notion called *miner-extractable value* (MEV) [90]. The term MEV is slightly deceptive: It does not refer *only* to value that miners can capture: Some MEV can be captured by ordinary users. Because miners have more power than ordinary users, however, MEV represents an upper bound on the amount of value any entity can obtain through adversarial reordering and complementary transaction insertion. Even when miners order transactions simply based on fees (gas), without manipulation, users themselves can manipulate gas prices to advantage their transactions over those of less sophistication.

Daian et al. [90] document and quantify ways in which bots (not miners) take advantage of gas dynamics in a way that harms users of DeFi systems today and how MEV even threatens the stability of the underlying consensus layer in a blockchain. Other examples of transaction-order manipulation surface regularly, e.g., [50, 154].

New transaction-processing methods such as rollups are a very promising approach to the scaling problems of high-throughput blockchains. They do not, however, address the problem of MEV. Instead, they shift it to the entity that generates the rollup. That entity, whether the operator of a smart contract or a user furnishing a (zk-)rollup with a validity proof, has the power to order and insert transactions. In other words, rollups swap MEV for REV: Rollup-Extractable Value.

MEV affects upcoming transactions that have been submitted to the mempool but are not yet committed on chain. Information about such transactions is broadly available in the network. Miners, validators, and ordinary network participants can therefore exploit this knowledge and create dependent transactions. In addition, miners and validators may influence the order of those transactions that they commit themselves and exploit this to their advantage.

The problem of undue influence by leaders on transaction ordering in consensus protocols has been known in the literature since the 1990s [71, 190], but no satisfying solutions have been realized in practice so far [97]. The main reason is that proposed solutions—at least until very recently—cannot readily be integrated with *public* blockchains, as they rely on the content of transactions remaining *secret* until *after* their ordering has been determined.

Fair Sequencing Services (FSS) overview: DONs will provide tools to decentralize transaction ordering and implement it according to a policy specified by a relying contract creator, ideally one that is *fair*, and not advantaging actors who wish to manipulate transaction ordering. Collectively, these tools constitute FSS.

FSS includes three components. The first is *monitoring* of transactions. In FSS, oracle nodes in $\mathcal{O}$ both monitor the mempool of MAINCHAIN and (if desired) permit off-chain submission of transactions through a specialized channel. The second is *sequencing* of transactions. The nodes in $\mathcal{O}$ order transactions for a relying contract according to a policy defined for that contract. The third is *posting* of transactions. After the transactions are ordered, the nodes in $\mathcal{O}$ jointly send the transactions to the main chain.

The potential benefits of FSS include:

- *Order-fairness:* FSS includes tools to help developers ensure that transactions input to a particular contract are ordered in a way that does not give an unfair advantage to well-resourced and/or technically savvy users. Ordering policies can be specified for this purpose.

- *Reduction or elimination of information leaks:* By ensuring that network participants cannot exploit knowledge about upcoming transactions, FSS can abate or eliminate attacks like front-running that are based on information available in the network before transactions are committed. Preventing exploitation of such leakage ensures that adversarial transactions which depend on original pending transactions cannot enter the ledger before the original transactions are committed.

- *Reduced transaction cost:* By eliminating players' need for speed in submitting their transactions to a smart contract, FSS can greatly reduce the cost of transaction processing.

- *Priority ordering:* FSS can automatically give critical transactions special priority ordering. For example, in order to prevent front-running attacks against oracle reports, e.g., [79], FSS can insert an oracle report into a stream of transactions retroactively.

An overarching goal of the FSS in DONs is to empower DeFi creators to realize *fair financial systems*, that is, systems that don't advantage particular users (or miners) over others on the basis of speed, insider knowledge, or ability to perform technical manipulation. While a crisp, general notion of fairness is elusive, and perfect fairness in any reasonable sense is unachievable, FSS aims to provide developers with a powerful set of tools so that they can enforce policies that help meet their design goals for DeFi.

We note that while the main goal of FSS is to act as a fair sequencing service for the MAINCHAIN that DONs target, some of the same fairness desiderata that FSS guarantees can also be appropriate for (decentralized) protocols that are run *among* DON parties. Thus, FSS can be viewed more broadly as a service provided by a subset of DON nodes to fairly sequence not only transactions sent by users of MAINCHAIN but also transactions (i.e., messages) shared among other DON nodes. In this section, we will focus primarily on the goal of sequencing MAINCHAIN transactions.

Section organization: In Section 5.1, we describe two high-level applications that motivate the design of FSS: preventing front-running of *oracle reports* and preventing front-running of *user transactions*. We then provide more details on the design of FSS in Section 5.2. Section 5.3 describes examples of fair ordering guarantees and means to achieve them. Finally, Section 5.4 and Section 5.5 discuss network-level threats to such policies and means to address them, respectively for *network flooding* and *Sybil attacks*.

5.1 The Front-Running Problem

To explain the goals and design of FSS, we describe two general forms of front-running attacks and the limitations of existing solutions. Front-running exemplifies a class of transaction-ordering attacks: There are a number of related attacks such as *back-running* and *sandwiching* (front-running plus back-running) [237] that we don't cover here, but which FSS also helps address.

5.1.1 Oracle Front-Running

In their traditional role of providing off-chain data to blockchain applications, oracles become a natural target for front-running attacks.

Consider the common design pattern of using an oracle to supply various *price feeds* to an on-chain exchange: periodically (say every hour), the oracle collects price data for different assets and sends these to an exchange contract. These price-data transactions present obvious arbitrage opportunities: For example, if the newest oracle report lists a much higher price for some asset, an adversary could front-run the oracle report to buy up assets and immediately resell them once the oracle's report is processed.

Speed bumps and retroactive pricing: A natural solution to the oracle front-running problem is to give oracle reports special *priority* over other transactions. For example, oracle reports could be sent with high fees to encourage miners to process them first. But this will not prevent front-running if the arbitrage opportunity is high, nor can it prevent arbitrage by the miners themselves.

Some exchanges have thus resorted to implementing more heavyweight "speed-bumps," such as *queuing* user transactions for a number of blocks before processing them, or *retroactively adjusting prices* when a new oracle report arrives. The disadvantages of these solutions are that they add complexity to the exchange implementation, increase storage requirements and thus transaction costs, and disrupt the user experience as asset exchanges are only confirmed after a significant time period.

Piggybacking: Before moving on to FSS, we discuss *piggybacking*, a quite simple and elegant solution to the oracle front-running problem. It is not applicable to address front-running in other scenarios, however.

In short, instead of periodically sending reports to the on-chain contract, oracles publish signed reports that users append to their transactions when buying or selling on-chain assets. The exchange then simply checks that the report is valid and fresh (e.g., the oracle can sign a range of blocks for which the report is valid), and extracts the relevant price feed from it.

This simple approach has a number of advantages over the above "speed bump" approach: (1) The exchange contract need not keep state of price feeds, which should lead to lower transaction costs; (2) As oracle reports are posted on chain on a by-need basis, oracles can generate more frequent updates (e.g., every minute), thereby minimizing arbitrage opportunities from front-running a report[9]; (3) Transactions can be validated immediately, as they always include a fresh price feed.

The approach is not perfect, however. First, this piggybacking solution puts the onus on the exchange's users to fetch up-to-date oracle reports and attach them to their transactions. Second, while piggybacking minimizes arbitrage opportunities, it cannot fully prevent them without affecting the liveness of the on-chain contract. Indeed, if an oracle report is valid until some block number n, then a transaction submitted to block $n+1$ would require a new valid report. Due to inherent delays in the propagation of reports from oracles to users, the new report that is valid for block $n+1$ would have

[9] Arbitrage is only worthwhile if the exploitable difference in asset prices exceeds the extraneous fees required to buy and sell the assets, e.g., those collected by miners and the exchange.

to be publicized some period before block $n+1$ is mined, say at block $n-k$, thereby creating a sequence of k blocks where a short-lived arbitrage opportunity exists. We now describe how FSS gets around these limitations.

Prioritizing oracle reports with FSS: FSS can address the oracle front-running problem by building upon the above piggybacking solution, but pushing the additional work of augmenting transactions with oracle reports to the Decentralized Oracle Network.

At a high level, oracle nodes collect transactions destined for an on-chain exchange, agree on a real-time price feed, and post the price feed along with the collected transactions to the main-chain contract. Conceptually, one can think of this approach as a "data-augmented transaction batching", where the oracle ensures that an up-to-date price feed is always added to transactions.

FSS solutions can be implemented transparently to the exchange's users, and with minimal changes to contract logic, as we describe in more detail in Section 5.2. Ensuring that fresh oracle reports are always prioritized over user transactions is just one example of an *ordering policy* that FSS can adopt and enforce. Policies of FSS for ensuring order fairness are described more generally in Section 5.3.

5.1.2 Front-Running User Transactions

We now turn to front-running in generic applications, where the defense method above does not work. The problem can be captured broadly through the following scenario: An adversary sees some user transaction tx_1 sent into the P2P network and injects its own adversarial transaction tx_2, so that tx_2 is processed before tx_1 (e.g., by paying a higher transaction fee). For instance, this kind of front-running is common among bots that exploit arbitrage opportunities in DeFi systems [90] and has affected users of various decentralized applications [101]. Imposing a fair order among the transactions processed on the blockchain addresses this problem.

More fundamentally, seeing the details of tx_1 is sometimes not even necessary and knowledge of its mere *existence* may allow an adversary to front-run tx_1 through its own tx_2 and defraud the innocent user that created tx_1. For example, the user might be known to trade a particular asset at regular times. Preventing such attacks requires mitigations that avoid leakage of metadata as well [62]. Some solutions for this problem exist, but they introduce delays and usability concerns.

From network-order to finalized-order with FSS: Opportunities for front-running arise because existing systems have no mechanisms to ensure that the order in which transactions appear on chain respects the order of events and the information flow outside the network. This represents a problem arising from deficiencies in the *implementation* of applications (e.g., trading platforms) on a blockchain. Ideally, one would ensure that transactions are committed on the blockchain in the same order as they were created and sent to the blockchain's P2P network. But since the blockchain network

is distributed, no such order can be captured. FSS therefore introduces mechanisms to safeguard against violations of fairness, which arise only because of the distributed nature of the blockchain network.

5.2 FSS Details

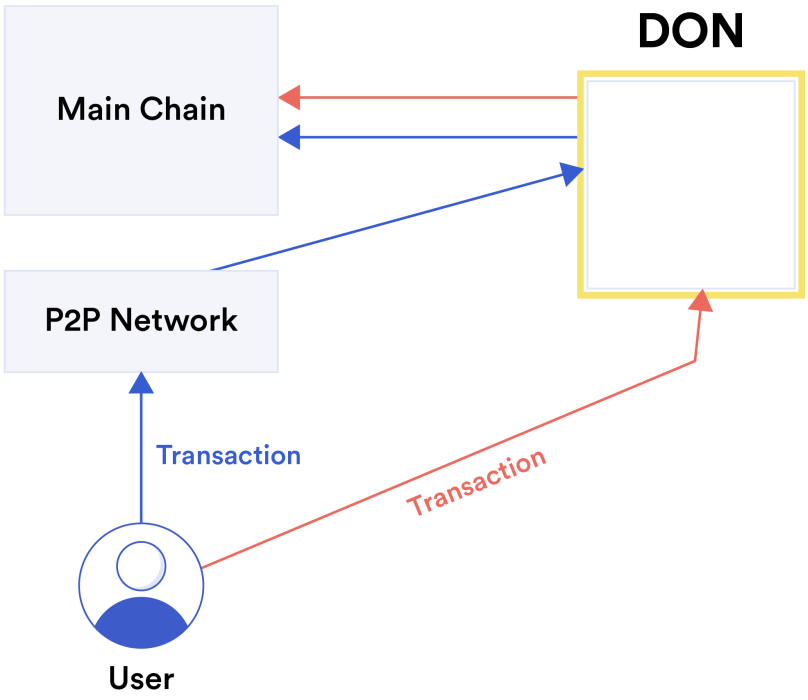

Figure 12: Order-fair mempool with two different transaction paths: direct and mempool-based.

Fig. 12 shows a general schematic of the FSS. For ensuring fairness, the DON providing FSS must interfere with the flow of transactions as they enter MAINCHAIN. Adjustments to clients, to smart contracts on MAINCHAIN, or to both may be necessary. At a high level, processing of transactions by FSS can be decomposed into three phases, described below: (1) Transaction monitoring; (2) Transaction sequencing; and (3) Transaction posting. Depending on the ordering method used for transaction sequencing, additional protocol steps are needed, as described in the next section.

5.2.1 Transaction Processing

Transaction monitoring: We envision two different approaches for FSS to monitor user transactions destined for a specific smart contract, *direct* and *mempool-based*:

- *Direct*: The direct approach is conceptually simplest, but requires changes to user clients so that transactions are sent directly to the Decentralized Oracle

Network nodes, rather than to the nodes of the main chain. The DON collects user transactions destined to a specific smart contract SC and orders them based on some ordering policy. The DON then sends the ordered transactions to the smart contract on the main chain. Some ordering mechanisms also require the direct approach because the user that creates a transaction must cryptographically protect it before sending it to FSS.

- *Mempool-based*: To facilitate the integration of FSS with legacy clients, the DON can use Mempool Services (MS) to monitor the main chain's mempool and collect transactions.

Direct transmission is likely to be the preferred implementation for many contracts, and we believe it should be fairly practical in many cases.

We briefly discuss how existing DApps could be *minimally* modified to support direct transmission *while preserving a good user experience*. We describe approaches using Ethereum and MetaMask [6] since these are the most popular choices today, but the mentioned techniques should extend to other chains and wallets. A recent Ethereum Improvement Proposal, "EIP-3085: Wallet add Ethereum chain RPC method" [100], will make it easy to target custom Ethereum chains (using a different `CHAIN_ID` than that of MAINCHAIN to prevent replay attacks) from MetaMask and other browser-based wallets. After implementation of this proposal, a DApp seeking to use a DON would simply add a single method call to their front-end to be able to directly transmit transactions to any DON exposing an Ethereum-compatible API. In the meantime, "EIP-712: Ethereum typed structured data hashing and signing" [49] provides a slightly more involved but already widely deployed alternative, where a DApp user can use MetaMask to sign structured data specifying a DON transaction. The DApp can send this signed structured data to the DON.

Finally, we note that hybrid approaches are also possible. For example, legacy clients can continue to send transactions into the main chain's mempool, but critical transactions (e.g., oracle reports) are sent to DON nodes directly (in particular, the set of nodes providing oracle reports such as price-feed updates and the set of nodes providing FSS may overlap or be identical).

Transaction sequencing: The main purpose of FSS is to guarantee that user transactions are ordered according to a pre-defined policy. The nature of this policy will depend on the application's needs and the types of unfair transaction orderings that it aims to prevent.

Since FSS on the DON is capable of processing data and maintaining local state, they may impose an arbitrary sequencing policy based on the information that is available at the oracles.

The particular ordering policies and their implementation are discussed subsequently in Section 5.3.

Transaction posting: After collecting and ordering user transactions, received either directly from users or collected from the mempool, the DON sends these transactions to the main chain. As such, a DON's interactions with the main chain remain subject to (potentially unfair) transaction ordering governed by the main chain's miners. To harness the benefits of decentralized transaction ordering, the target smart contract SC thus has to be designed to treat the DON as a "first-class" citizen. We distinguish two approaches:

- *DON-only contracts:* The simplest design option is to have the main chain smart contract SC only accept transactions that have been processed by the DON. This ensures that the smart contract processes transactions in the order proposed by the DON, but de-facto restricts the smart contract to operating in a committee-based system (i.e., the DON committee now has ongoing power to determine the ordering and inclusion of transactions).

- *Dual-class contracts:* A preferred, more granular design has the main chain smart contract SC accept transactions originating both from the DON and from legacy users,[10] but places traditional "speed bumps" on transactions that were not processed by the DON. For example, transactions from the DON may be processed immediately, whereas legacy transactions get "buffered" by the smart contract for a fixed period of time. Other standard mechanisms for preventing front-running such as commit-reveal schemes or VDFs [53] could also be applied to legacy transactions. This ensures that DON-ordered transactions do get processed in the order agreed upon, without giving the DON the unwanted power to censor transactions.

As the imposition of transaction ordering by FSS requires that transactions are aggregated "off-chain," this solution is naturally combined with other aggregation techniques that aim to reduce on-chain processing costs. For example, after collecting and ordering transactions, the DON may send these transactions to the main chain as a single "batched transaction" (e.g., a rollup), thereby reducing the aggregate transaction fee.

Enforcing the transaction order: Whether in a DON-only or dual-class design, the main chain smart contract SC and the DON have to be co-designed so as to guarantee that the DON's transaction ordering is upheld. Here also, we envision different design options:

- *Sequence numbers:* The DON can append a sequence number to each transaction, and send these transactions into the main chain's mempool. The main

[10] If the DON's transaction monitoring is based on the mempool, legacy transactions must be distinguishable from DON transactions so that they are not collected by the DON, e.g., via a special tag embedded in the transaction or by specifying a particular gas price, e.g. DON transactions have gas prices below a certain threshold.

chain smart contract SC ignores transactions that arrive "out-of-sequence." We note that in this setting, the main-chain miners can decide to ignore the DON's transaction ordering, thereby causing transactions to fail. It is possible by keeping (expensive) state for SC to enforce correct transaction ordering, somewhat analogously to how TCP buffers out-of-order packets until missing packets are received.

- *Transaction nonces:* For many blockchains, and in particular for Ethereum, the above sequence-numbering approach can leverage built-in *transaction nonces* to enforce that the main-chain smart contract SC processes transactions in sequence. Here, the DON nodes send transactions to the main chain through a single main-chain account, protected with a key shared among the DON nodes. The account's transaction nonce ensures that transactions are mined and processed in the correct order.

- *Aggregate transactions:* The DON can aggregate multiple transactions in a rollup (or in a bundle similar to a rollup). The main-chain smart contract needs to be designed to handle such aggregate transactions.

- *Aggregate transactions with a main chain proxy:* Here, the DON similarly bundles transactions into one "meta-transaction" for the main chain, but relies on a custom proxy smart contract to unpack the transactions and relay them to the target contract SC. This technique can be useful for legacy compatibility. Meta-transactions act like rollups but differ in that they consist of an uncompressed list of transactions posted once to the main chain.

The last design has the advantage of seamlessly supporting user transactions that are themselves proxied through a main chain contract before reaching the DON's target contract SC. For example, consider a user who sends a transaction to some wallet contract, which in turn sends an internal transaction to SC. Assigning a sequence number to such a transaction would be tricky, unless the user's wallet contract is specially designed to forward the sequence number with every internal transaction to SC. Similarly, such internal transactions cannot be easily aggregated into a meta-transaction that is sent directly to SC. We discuss further design considerations for such proxied transactions below.

5.2.2 Transaction Atomicity

Our discussion thus far has implicitly assumed that transactions interact with a single on-chain smart contract (e.g., a user sends a buy request to an exchange). Yet, in systems such as Ethereum, a single transaction can consist of multiple *internal* transactions, e.g., one smart contract calling a function in another contract. Below, we describe two high-level strategies for sequencing "multi-contract" transactions, while preserving the *atomicity* of the transaction (i.e., the sequence of actions prescribed by the transaction are all executed in the correct order, or not at all).

Strong atomicity: The simplest solution is to apply FSS, as described above, directly to entire "multi-contract" transactions. That is, users send their transactions into the network and FSS monitors, sequences, and posts these transactions to the main chain.

This approach is technically simple, but has one potential limitation: If a user transaction interacts with two contracts SC_1 and SC_2 that both want to leverage fair sequencing services, then the *sequencing policy* of these two contracts has to be *consistent*. That is, given two different transactions tx_1 and tx_2 that each interacts with both SC_1 and SC_2, it must not be the case that the policy of SC_1 orders tx_1 before tx_2 whereas the policy of SC_2 prescribes the opposite order.

For the vast majority of scenarios of interest, we envision that the sequencing policies adopted by different contracts will be consistent. For example, both SC_1 and SC_2 may want transactions to be ordered by their approximate arrival time in the mempool, and SC_1 may further want certain oracle reports to always be delivered first. As the latter oracle report transactions do not interact with SC_2, the policies are consistent.

Weak atomicity: In its full generality, FSS could be applied at the level of individual *internal* transactions.

Consider transactions of the form $\mathsf{tx} = \{\tilde{\mathsf{tx}}_{\text{pre}}, \tilde{\mathsf{tx}}_{\mathsf{SC}}, \tilde{\mathsf{tx}}_{\text{post}}\}$, consisting of some initial transaction(s) $\tilde{\mathsf{tx}}_{\text{pre}}$, which results in an internal transaction $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ to SC, which in turn issues internal transaction(s) $\tilde{\mathsf{tx}}_{\text{post}}$. The sequencing policy of SC might determine how the internal transaction $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ has to be ordered with respect to other transactions sent to SC, but leave open the sequencing order for $\tilde{\mathsf{tx}}_{\text{pre}}$ and $\tilde{\mathsf{tx}}_{\text{post}}$.

Given the intrinsics of transaction processing in systems such as Ethereum, developing a sequencing service that targets specific internal transactions is not straightforward. With a specially designed contract SC, this may be realizable as follows:

1. The transaction tx is sent into the network and mined (without any sequencing performed by FSS). The initial $\tilde{\mathsf{tx}}_{\text{pre}}$ is executed, and calls $\tilde{\mathsf{tx}}_{\mathsf{SC}}$.

2. SC does not execute $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ and returns.

3. FSS monitors internal transactions to SC, sequences them, and posts them back to SC (i.e., by sending transactions $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ directly to SC).

4. SC processes the transactions $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ received from FSS, and issues internal transactions $\tilde{\mathsf{tx}}_{\text{post}}$ that result from $\tilde{\mathsf{tx}}_{\mathsf{SC}}$.

With this approach, transactions are not executed fully atomically (i.e., the original transaction tx gets broken up into multiple on-chain transactions), but the ordering of internal transactions is preserved.

This solution entails a number of design constraints. For example, $\tilde{\mathsf{tx}}_{\text{pre}}$ cannot assume that $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ and $\tilde{\mathsf{tx}}_{\text{post}}$ will be executed. Moreover, SC should be designed so as to execute transactions $\tilde{\mathsf{tx}}_{\mathsf{SC}}$ and $\tilde{\mathsf{tx}}_{\text{post}}$ on behalf of a certain user, even though they were

sent by FSS. For these reasons, the more coarse-grained "Strong Atomicity" solution above is likely preferable in practice.

For respecting more complex dependencies, involving multiple transactions and their respective internal transactions, the transaction scheduler of FSS may contain elaborate functions that resemble those found in transaction managers of relational database managers.

5.3 Fair Transaction Sequencing

Here we discuss two notions of fairness for transaction sequencing and the corresponding implementations, which may be realized by FSS: *order-fairness* based on a policy imposed by FSS and *secure causality preservation*, which requires additional cryptographic methods in FSS.

Order-fairness: Order-fairness is a notion of *temporal* fairness in consensus protocols that has first been introduced formally by Kelkar et al. [144].

Kelkar et al. aim to achieve a form of natural policy in which transactions are ordered based on the time they are first received by the DON (or the P2P network, in the case of a mempool-based FSS). In a decentralized system, however, different nodes may see transactions arrive in a different order. Establishing a total order on all transactions is the very problem solved by the consensus protocol underlying MAINCHAIN. Kelkar et al. [144] therefore introduce a weaker notion that can be achieved with the help of a Decentralized Oracle Network, called "block-order fairness." It groups the transactions that the DON has received during a time interval into a "block" and inserts all transactions of the block concurrently and at the same position (i.e., height) into MAINCHAIN. They are thus ordered together and must be executable in parallel, without creating any conflicts among them. Roughly speaking, order-fairness then states that if a large fraction of nodes see transaction τ_1 before τ_2, then τ_1 will be sequenced before or in the same block as τ_2. By imposing such a coarse granularity on transaction order, the opportunities for front-running and other order-related attacks are greatly reduced.

Kelkar et al. propose a family of protocols called *Aequitas* [144], which address different deployment models, including synchronous, partially synchronous, and asynchronous network settings. Aequitas protocols impose significant communication overhead relative to basic BFT consensus and are therefore not ideal for practical use. We believe, however, that practical variants of Aequitas will emerge that can be used for transaction sequencing in FSS and other applications. Some related schemes have already been proposed that have less accompanying formalism and weaker properties, e.g., [36, 151, 236], but better practical performance. These schemes can be supported in FSS as well.

It is also worth noting that the term "fairness" appears elsewhere in the blockchain literature with a different meaning, namely fairness in the sense of opportunity for

miners proportional to their committed resources [106, 181] or for validators in terms of equal opportunity [153].

Secure causality preservation: The most widely known approach to prevent front-running and other ordering violations in distributed platforms relies on cryptographic techniques. Their common feature is to hide the transaction data itself, waiting until the order at the consensus layer has been established, and to reveal the transaction data later for processing. This preserves the *causal order* among the transactions that are executed by the blockchain. The relevant security notions and cryptographic protocols have been developed considerably before the advent of blockchains [71, 190].

The security conditions of "input causality" [190] and "secure causality preservation" [71, 97] require formally that no information about a transaction becomes known before the position of this transaction in the global order has been determined. An adversary must not be able to infer any information until that time, in a cryptographically strong sense.

One can distinguish four cryptographic techniques to preserve causality:

- *Commit-reveal protocols [29, 142, 145]:* Instead of a transaction being announced in the clear, only a *cryptographic commitment* to the transaction is broadcast. After all committed but hidden transactions have been ordered (in early blockchain systems on MAINCHAIN itself, but here by FSS), the sender must open the commitment and reveal the transaction data within a predetermined time interval. The network then verifies that the opening satisfies the earlier commitment. The origins of this method date prior to the advent of blockchains.

  Although it is particularly simple, the approach introduces considerable drawbacks and is not easy to employ for two reasons. First, since only the commitment exists at the level of the ordering protocol, the semantics of the transaction cannot be validated during consensus. An additional round-trip to the client is required. More severely, though, weighs the possibility that no opening may ever arrive, which could amount to a denial-of-service attack. Furthermore, it is difficult to determine whether the opening is valid in a consistent, distributed manner because all participants must agree on whether the opening arrived in time.

- *Commit-reveal protocols with delayed recovery [145]:* One challenge with the commit-reveal approach is that a client may commit to a transaction speculatively and reveal it later only if subsequent transactions make it profitable. A recent variant of the commit-reveal approach improves the resilience against this kind of misbehavior. In particular, the TEX protocol [145] addresses this problem using a clever approach in which encrypted transactions include a decryption key obtainable by computing a verifiable delay function (VDF) [53, 221]. If a client fails to decrypt her transaction in a timely way, others in the system will decrypt it on her behalf by solving a moderately hard cryptographic puzzle.

- *Threshold encryption [71, 190]:* This method exploits that the DON may perform threshold-cryptographic operations. Assume FSS maintains an encryption public key $\mathsf{pk}_{\mathcal{O}}$ and the oracles share the corresponding private key among themselves. Clients then encrypt transactions under $\mathsf{pk}_{\mathcal{O}}$ and send them to FSS. FSS orders transactions on the DON, then decrypts them, and finally injects them into MAINCHAIN in the fixed order. Encryption therefore ensures that ordering is not based on the transaction content, but that the data itself is available when needed.

  This method was originally proposed by Reiter and Birman [190] and later refined by Cachin et al. [71], where it was integrated with a permissioned consensus protocol. More recent work has explored the use of threshold cryptography as a consensus-level mechanism for generic messages [33, 97] and for general computations with shared data [41].

  Compared to commit-reveal protocols, threshold encryption prevents simple denial-of-service attacks (although care is required given the computational cost of decryption). It lets the DON proceed autonomously, at its own speed and without waiting for further client actions. Transactions may be validated immediately after they have been decrypted. Moreover, clients encrypt all transactions with one key for the DON and the communication pattern remains the same as with other transactions. Managing the threshold key securely and with changing nodes in $\mathcal{O}$, however, may pose additional difficulties.

- *Committed secret sharing [97]:* Instead of encrypting the transaction data under a key held by the DON, the client may also secret-share it for the nodes in $\mathcal{O}$. Using a hybrid, computationally secure secret sharing scheme, the transaction is encrypted first using a symmetric cipher with a random key. Only the corresponding symmetric key is shared and the ciphertext is submitted to the DON. The client must send one key share to each node in $\mathcal{O}$ using a separately encrypted message. The remaining protocol steps are the same as with threshold encryption, except that the transaction data is decrypted with the symmetric algorithm after reconstructing the per-transaction key from its shares.

  This method does not require setup or management of a public-key cryptosystem associated with the DON. However, the clients must be aware of the nodes in $\mathcal{O}$ and communicate in a secure context with each one of them, which places additional burden on the clients.

Although the cryptographic methods offer complete protection against information leaking from submitted transactions to the network, they do not conceal *metadata*. For example, an IP address or an Ethereum address of the sender could still be used by an adversary to perform front-running and other attacks. Various privacy-enhancing techniques deployed at the network layer, e.g., [52, 95, 107], or the transaction layer, e.g., [13, 65], would be needed to accomplish this goal. The impact of a particular piece of metadata, namely to which *contract* a transaction is sent, can be (partially) concealed

through multiplexing many contracts on the same DON. Cryptographic concealment of transactions *per se* also doesn't prevent prioritization of transactions by corrupted DON nodes in collusion with transaction senders.

Secure causality as guaranteed by cryptographic protocols complement the order-fairness guarantees for any policy, and we intend to explore a combination of the two methods, where this is possible. If an adversary cannot gain significant advantage from observing metadata, the secure causality-preservation protocols could be used alongside a naïve ordering approach as well. For example, oracle nodes can write transactions to $\mathcal{L}$ as soon as they receive them, without duplication. Transactions would then be ordered according to their appearance on $\mathcal{L}$ and subsequently decrypted.

We also plan to consider the use of TEEs as a way to help enforce fair ordering; for example, Tesseract [44] may be viewed as achieving a form of causal ordering, but one strengthened by the ability of the TEE to process transactions in explicit form while retaining their confidentiality.

5.4 Network-Layer Considerations

So far, our description of FSS has mainly focused on the problem of enforcing that the finalized order of transactions matches their observed order in the network. Hereafter, we consider fairness issues that could arise at the network layer itself.

High-frequency traders in conventional electronic marketplaces invest considerable resources to obtain superior network speed [64], and traders in cryptocurrency exchanges exhibit similar behavior [90]. Network speed confers an advantage both in observing the transactions of other parties and in submitting competing transactions. One remedy deployed in practice and popularized in the book *Flash Boys* [155] is the "speed bump" introduced initially in the IEX exchange [128] and later in other exchanges [179] (with mixed results [19]). This mechanism imposes a delay (350 microseconds in IEX) on access to the market, with the aim of neutralizing advantages in speed. Empirical evidence, e.g. [128], supports its efficacy in decreasing certain trading costs for ordinary investors. FSS can be used simply to implement an asymmetrical speed bump—one that delays incoming transactions.

Budish, Cramton, and Shim [64] argue that exploitation of advantages in speed is inescapable in continuous-time markets, and argue for a structural remedy in the form of batch-auction-based markets. But this approach has not taken hold broadly in existing trading platforms.

Conventional trading systems are centralized, typically receiving transactions through a single network connection. In a decentralized system, by contrast, it is possible to observe transaction propagation from multiple vantage points. Consequently, it is possible to observe behaviors such as network flooding in a P2P network. We intend to explore network-layer approaches to FSS that help developers to specify policies prohibiting such undesirable network behaviors.

5.5 Entity-Level Fairness Policies

Order-fairness and secure causality aim at enforcing an ordering on transactions that respects the time when they were created and first submitted to the network. A limitation of this notion of fairness is that it does not prevent attacks in which an adversary gains an advantage by flooding a system with many transactions, a strategy observed in the wild as a way to perform effective transaction sniping in token sales [159] and to create congestion resulting in liquidation of collateralized debt positions (CDPs) [48]. In other words, order-fairness enforces fairness *with respect to transactions, not players.*

As shown in the CanDID system [160], it is possible to use oracle tools such as DECO or Town Crier in conjunction with a committee of nodes (such as a DON) to achieve various forms of *Sybil-resistance* while protecting privacy. Users can register identities and provide evidence of their uniqueness without disclosing the identities themselves. Sybil-resistant credentials offer a possible approach to enriching transaction-ordering policies in a way that would limit opportunities for flooding attacks. For example, a token sale might permit only one transaction per registered user, where registration requires a proof of uniqueness of a national identifier, such as a Social Security Number. Such an approach isn't foolproof, but may prove a useful policy to mitigate transaction-flooding attacks.

6 The DON Transaction-Execution Framework (DON-TEF)

DONs will provide oracle and decentralized-resource support for layer-2 solutions within what we call the *Decentralized Oracle Network Transaction-Execution Framework* (DON-TEF) or TEF for short.

Today, the frequency of updates to DeFi contracts is limited by main chain latencies, e.g., the 10-15 second average block interval in Ethereum [104]—as well as the cost of pushing large amounts of data on chain and limited computational/tx throughput—motivating scaling approaches such as sharding [148, 158, 232] and layer-2 execution [5, 12, 121, 141, 169, 186, 187]. Even blockchains with much faster transaction times, e.g., [120], have proposed scaling strategies that involve off-chain computation [168]. TEF is meant to act as a layer-2 resource for any such layer-1 / MAINCHAIN systems.

Using TEF, DONs can support faster updates in a MAINCHAIN contract while retaining the key trust assurances provided by the main chain. TEF can support any of a number of layer-2 execution techniques and paradigms, including rollups,[11] optimistic rollups, Validium, etc., as well as a threshold trust model in which DON nodes execute transactions.

The TEF is complementary to FSS and intended to support it. In other words, any application running in the TEF can use FSS.

[11] Often called "zk-rollups," a misnomer, as they do not necessarily need zero-knowledge proofs.

6.1 TEF Overview

The TEF is a design pattern for the construction and execution of a performant hybrid smart contract SC.

In accordance with the main idea behind hybrid smart contracts, TEF involves a decomposition of SC into two pieces: (1) What we call in the TEF context an *anchor* contract SC_a on MAINCHAIN and (2) DON logic exec_t that we call the TEF executable. We use SC here to denote the logical contract implemented by the combination of SC_a and exec_t. (As noted above, we expect to develop compiler tools to decompose a contract SC automatically into these components.)

The TEF executable exec_t is the engine that processes users' transactions in SC. It can execute in a performant way, as it runs on the DON. It has several functions:

- *Transaction ingestion:* exec_t receives or fetches users' transactions. It can do so directly, i.e., through transaction submission on the DON, or via the MAINCHAIN mempool using MS.
- *Fast transaction execution:* exec_t processes transactions involving assets within SC. It does so locally, i.e., on the DON.
- *Fast and low-cost oracle / adapter access:* exec_t has native access to oracle reports and other adapter data leading to, e.g., faster, cheaper, and more accurate asset pricing than MAINCHAIN execution. Moreover, off-chain oracle access reduces the oracle's operational cost, hence the cost of using the system, by avoiding expensive on-chain storage.
- *Syncing:* exec_t periodically pushes updates from DON onto MAINCHAIN, updating SC_a.

The anchor contract is the MAINCHAIN front end of SC. As the higher-trust component of SC, it serves several purposes:

- *Asset custody:* Users' funds are deposited into, held in, and withdrawn from SC_a.
- *Syncing verification:* SC_a may verify the correctness of state updates when exec_t syncs, e.g., SNARKs attached to rollups.
- *Guard rails:* SC_a may include provisions to protect against corruption or failures in exec_t. (See Section 7 for more details.)

In TEF, users' funds are custodied on MAINCHAIN, meaning the DON is itself non-custodial. Depending on the choice of syncing mechanism (see below), users may need to trust the DON only for accurate oracle reports and timely syncing with MAINCHAIN. The resulting trust model is very similar to that for order-book-based DEXes, e.g., [2], which today generally include an off-chain component for order matching and an on-chain component for clearing and settlement.

To use the vocabulary of payment systems, one may think of exec_t as the component of SC responsible for *clearing*, while SC_a handles *settlement*. See Fig. 13 for a schematic depiction of TEF.

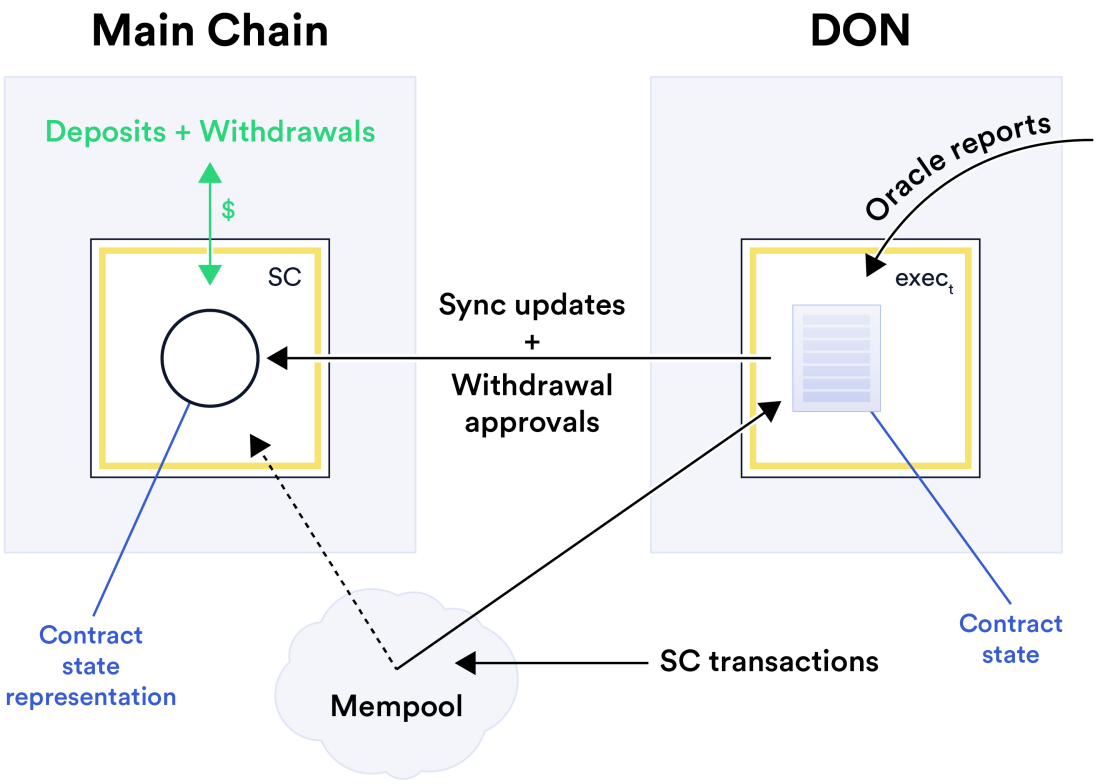

Figure 13: TEF schematic. In this example, transactions pass through the mempool of MAINCHAIN via MS to the DON.

TEF benefits: TEF carries three main benefits:

- *High performance:* SC inherits the DON's much higher throughput than MAINCHAIN for both transactions and oracle reports. Additionally, exec_t can process transactions faster and respond to oracle reports in a more timely way than an implementation on MAINCHAIN alone.
- *Lower fees:* The process of syncing is less time-sensitive than transaction processing, and transactions can be sent from the DON to MAINCHAIN in batches. Consequently, the per-transaction on-chain fees (e.g., gas costs) with this approach are much lower than for a contract that runs only on MAINCHAIN.
- *Confidentiality:* The confidentiality mechanisms of the DON can be brought to bear on SC.

TEF limitations: One limitation of TEF is that it does not support instantaneous withdrawals, as they occur only on MAINCHAIN: Upon sending a withdrawal request to SC_a, a user may need to wait for exec_t to perform a state update that includes the withdrawal transaction before it can be approved. We discuss some partial remedies, however, in Section 6.2.

Another limitation of TEF is that it does not support atomic composition of DeFi contracts on MAINCHAIN, specifically the ability to route assets through multiple DeFi contracts in a single transaction. TEF *can*, however, support such atomicity among DeFi contracts running on the same DON. We also discuss some ways to address this problem in Section 6.2.

6.2 Transaction Routing

Transactions for SC can be sent by users directly to the DON or can be routed through the mempool in MAINCHAIN (via FSS). There are four distinct transaction types, each of which requires different handling:

Within-contract transactions: Because it sidesteps the complications of gas dynamics, TEF provides SC more flexibility in its handling of transactions than would be available in a layer-1 contract. For example, while a mempool transaction in Ethereum can be overwritten by a fresh transaction with a higher gas price, SC can treat a transaction that operates on assets within SC as authoritative as soon as it becomes visible in the mempool. Consequently, SC need not wait for a transaction to be confirmed within a block, resulting in considerably reduced latency.

Proxying: A user may wish to send a transaction τ to SC via a wallet contract or other contract on MAINCHAIN. It is possible for the DON to *simulate* execution of τ on MAINCHAIN to determine whether it results in a follow-on transaction to SC. If so, τ can be sequenced with other transactions for SC that do. There are a few possibilities for how the DON identifies such transactions: (1) The DON can simulate all transactions in the mempool (an expensive approach); (2) Certain contracts or contract types, e.g., wallets, can be listed for monitoring by the DON; or (3) Users can annotate transactions for DON inspection.

Matters become more complicated when a single transaction interacts with two contracts, SC_1 and SC_2, both of which use Fair Sequencing Services and have incompatible ordering policies. The DON might, for example, sequence τ at the latest time that is compatible with both.

Deposits: A transaction depositing a MAINCHAIN asset into SC needs to be confirmed in a block before SC can treat it as valid. When it detects the mining of a transaction that sends assets (e.g., Ether) into SC_a, exec_t can instantly confirm the

deposit. For example, it can apply a current oracle-reported price on the DON to the asset.

Withdrawals: As noted above, a limitation of TEF is that withdrawals cannot always be executed instantaneously. In a rollup-type execution model, the withdrawal request must be sequenced with other transactions, i.e., rolled up, in order to be safely processed. There are, however, some partial remedies to this limitation.

If the DON can quickly compute a rollup validity proof up to the withdrawal transaction, then observing a user's transaction τ in the mempool exec_t can send a state-update transaction τ' for τ at a higher gas price, a kind of *beneficial front-running*. Provided that τ isn't mined before τ' reaches the mempool, τ' will precede τ, and τ will effect an approved withdrawal.

In a TEF variant where the DON is relied upon to compute state updates (see the threshold signing variant below), the DON can alternatively determine off-chain whether τ ought to be approved given the state of SC upon its execution. The DON can then send a transaction τ' that approves withdrawal τ—without effecting a full state update.

If this approach isn't possible, or in cases where it doesn't succeed, a DON-initiated transaction τ' can send funds to the user in response to τ so that the user need not initiate an additional transaction.

6.3 Syncing

The TEF executable exec_t periodically pushes updates from DON to MAINCHAIN, updating the state of SC_a in a process we refer to as *syncing*. Syncing may be thought of as propagation of layer-2 transactions to layer-1, so TEF can draw on any of a number of existing techniques for this purpose, including rollups [5, 12, 16, 69], optimistic rollups [10, 11, 141], Validium [201], or basic threshold signing, e.g., threshold BLS, Schnorr, or ECDSA [24, 54, 116, 202]. In principle, trusted execution environments can also attest to the correctness of state changes, offering a much more performant alternative to rollups, but with a hardware-dependent trust model. (See, e.g., [80].)

Below we compare these syncing options with respect to three key properties in TEF:

- *Data availability*: Where is the state of SC stored? At least three options are available in TEF: on the MAINCHAIN, on a DON, or by some third-party storage providers such as IPFS. They achieve different security guarantees, availability levels, and performance profiles. Briefly, storing state on the MAINCHAIN enables on-chain auditability and eliminates reliance on any party for state availability; on the other hand, storing state off-chain can reduce storage cost and improve throughput, at the cost of trusting storage providers (DON or third parties) for data availability. Of course, flexible models that combine these options are also possible. We indicate the required form of data availability in Table 1.

- *Correctness guarantees*: How does SC_a ascertain the correctness of the updates pushed by exec_t? This affects the computational load on exec_t and SC_a and the syncing latency (see below).
- *Latency*: Syncing latency has three contributing factors: (1) The time taken for exec_t to generate a syncing transaction τ_{sync}; (2) The time taken for τ_{sync} to be confirmed on MAINCHAIN; and (3) The time for τ_{sync} to take effect on SC_a. In TEF, latency is particularly important for withdrawals (but less so for within-contract transactions) because withdrawals necessarily require an (at least partial) state sync.

Syncing options	**Data availability**	**Correctness guarantees**	**Latency**
Rollup [5, 12, 16, 69]	On-chain	Validity proofs	Time taken to generate validity proofs (e.g., minutes in current systems)
Validium [201]	Off-chain	Validity proofs	Same as above
Optimistic rollup [10, 11, 141]	On-chain	Fraud proofs	Length of the challenge period (e.g., days or weeks)
Threshold signing [24, 54, 116, 202]	Flexible	Threshold signatures by DON	Instantaneous
Trusted execution environments [80]	Flexible	Hardware-based attestations	Instantaneous

Table 1: Various syncing options in TEF and their properties.

Table 1 summarizes these properties in the five main syncing options in TEF. (Note that we do not intend to compare these technologies as *standalone layer-2 scaling solutions.* For that we refer readers to e.g., [121].)

Now we discuss each syncing option.

Rollups: A rollup [69] is a protocol in which the state transition effected by a batch of transactions is computed off-chain. The state change is then propagated onto MAINCHAIN.

To implement rollups, the anchor smart contract SC_a stores a compact representation R_{state} (e.g., a Merkle root) of the actual state. To sync, exec_t sends $\tau_{\text{sync}} = (T, R'_{\text{state}})$ to SC_a where T is the set of the transactions it processed since the last

sync and R'_{state} is the compact representation of the new state calculated by applying transactions in T to the previous state R_{state}.

There are two popular variants that differ in how SC_a verifies state updates in τ_{sync}. The first, **(zk-)rollups**, attach a succinct argument of correctness, sometimes called a *validity proof*, for the transition $R_{\text{state}} \to R'_{\text{state}}$. To implement this variant, exec_t computes and submits the validity proof (e.g., a zk-SNARK proof) along with τ_{sync}, proving that R'_{state} is the result of applying T to the current state of SC_a. The anchor contract accepts the state update only after it has verified the proof.

Optimistic rollups do not include arguments of correctness, but have staking and challenge procedures that facilitate distributed verification of state transitions. For this rollup variant, SC_a tentatively accepts τ_{sync} assuming it is correct (hence the optimism) but τ_{sync} does not take effect until after a challenge period, during which any party monitoring MAINCHAIN can identify erroneous state updates and inform SC_a to take necessary actions (e.g., to rollback the state and inflict a penalty on exec_t.)

Both rollup variants achieve on-chain data availability, as transactions are posted on-chain, from which the full state can be constructed. The latency of zk-rollups is dominated by the time needed to generate validity proofs, which typically is on the order of minutes in existing systems [16] and will likely see improvements over time. Optimistic rollups, on the other hand, have a higher latency (e.g., days or weeks) because the challenge period needs to be long enough for fraud proofs to work. The implication of slow confirmation is subtle and sometimes specific to the scheme, so that a thorough analysis is out of scope. For instance, certain schemes consider payment transactions as "trustless final" [109] before the state update is confirmed, since a regular user could verify a rollup much more quickly than the MAINCHAIN.

Validium: Validium is a form of (zk-)rollup that makes data available off-chain only and does not maintain all data on MAINCHAIN. Specifically, exec_t sends only the new state and the proof but not transactions to SC_a. With Validium-style syncing, exec_t and the DON that executes it are the only parties that store the complete state and that execute transactions. As with zk-rollups, syncing latency is dominated by validity proof generation time. Unlike zk-rollups, however, Validium style syncing reduces the storage cost and increases the throughput.

Threshold signing by DON: Assuming a threshold of DON nodes is honest, a simple and fast syncing option is to have DON nodes collectively sign the new state. This approach can support both on-chain and off-chain data availability. Note that if users trust DON for oracle updates, they do not need to trust it more for accepting state updates, as they are already in a threshold trust model. Another benefit of threshold signing is low latency. Support for new transaction signature formats as proposed in EIP-2938 [70] and known as *account abstraction* would make threshold signing considerably easier to implement, as it would eliminate the need for threshold ECDSA, which involves considerably more complex protocols (e.g., [116, 117, 118])

than alternatives such as threshold Schnorr [202] or BLS [55] signatures.

Trusted Execution Environments (TEEs): TEEs are isolated execution environments (usually realized by hardware) that aim to provide strong security protections for programs running inside. Some TEEs (e.g., Intel SGX [84]) can produce proofs, known as *attestations*, that an output is correctly computed by a specific program for a particular input[12]. A TEE-based variant of TEF syncing can be implemented by replacing proofs in (zk-)rollups or Validium with TEE attestations using techniques from [80].

Compared to zero-knowledge proofs used in rollups and Validium, TEEs are much more performant. Compared to threshold signing, TEEs remove the complexity of generating threshold ECDSA signatures as there need in principle be only one TEE involved. Using TEEs does, however, introduce extra hardware-dependent trust assumptions. One can also combine TEEs with threshold signing to create resilience against compromise of a fraction of TEE instances, although this protective measure reintroduces the complexity of generating threshold ECDSA signatures.

Additional flexibility: These syncing options can be refined to provide more flexibility in the following ways.

- *Flexible triggering:* TEF application can determine the conditions under which syncing is triggered. For example, syncing can be batch-based, e.g., occur after every N transactions, time-based, e.g., every 10 blocks, or event-based, e.g., occur whenever target asset prices move significantly.

- *Partial syncing:* It is possible and in some cases desirable (e.g., with rollups, partial syncing can reduce latency) for exec_t to provide fast syncing of small amounts of state, performing full syncing perhaps only periodically. For example, exec_t can approve a withdrawal request by updating a user's balance in SC_a without otherwise updating MAINCHAIN state.

6.4 Reorgs

Blockchain reorganizations resulting from network instability or even from 51%-attacks can pose a threat to the integrity of a main chain. In practice, adversaries have used them to mount double-spending attacks [34]. While such attacks on major chains are challenging to mount, they remain feasible for some chains [88].

Because it operates independently of MAINCHAIN, a DON offers the interesting possibility of observing and providing some protections against reorgs associated with attacks.

For example, a DON can report to a relying contract SC on MAINCHAIN the existence of a competing fork of some threshold length τ. The DON can additionally

[12]Supplementary details can be found in Appendix B.2.1. They are not required for understanding.

provide proof—in either a PoW or PoS setting—of the existence of such a fork. The contract SC can implement suitable defensive actions, such as suspending further transaction execution for a period of time (e.g., to allow exchanges to blacklist double-spent assets). Note that although an adversary mounting a 51%-attack can seek to censor reports from a DON, a countermeasure in SC is to require periodic reports from the DON in order to process transactions (i.e., a heartbeat) or to require a fresh report to validate a high-value transaction.

While such forking alerts are in principle a general service the DON can provide for any of a number of purposes, our plan is to incorporate them with the TEF.

7 Trust Minimization

As a decentralized system with participation from a heterogeneous set of entities, the Chainlink network provides strong protection against failures in both liveness (availability) and safety (report integrity). Most decentralized systems, however, vary in the degree to which their constituent components are themselves decentralized. This is true even of large systems, where limited decentralization among miners [32] and intermediaries [51] has long been present.

The goal of any decentralization effort is *trust minimization*: We seek to reduce the adverse effects of systemic corruption or failure within the Chainlink network, even that due to a malicious DON. Our guiding principle is the *Principle of Least Privilege* [197]. System components and actors within the system should have privileges strictly scoped to allow *only* for the successful completion of their assigned roles.

Here we lay out several concrete mechanisms for Chainlink to adopt in its drive toward ever-greater trust minimization. We characterize these mechanisms in terms of the loci, i.e., system components, in which they are rooted, shown in Fig. 14. We address each locus in a respective subsection.

7.1 Data-Source Authentication

Current operating models for oracles are constrained by the fact that few data sources digitally sign the data they omit, in large part because TLS does not natively sign data. TLS does make use of digital signatures in its "handshake" protocol (to establish a shared key between a server and client). HTTPS-enabled servers thus have certificates on public keys that can in principle serve to sign data, but they do not generally use these certificates to support data signing. Consequently, the security of a DON, as in today's oracle networks, relies on oracle nodes faithfully relaying data from a data source to a contract.

An important long-term component of our vision for trust minimization in Chainlink involves stronger data-source authentication through support of tools and standards for data signing. Data signing can help enforce *end-to-end* integrity guarantees. In principle, if a contract accepts as input a piece of data D signed directly by a data

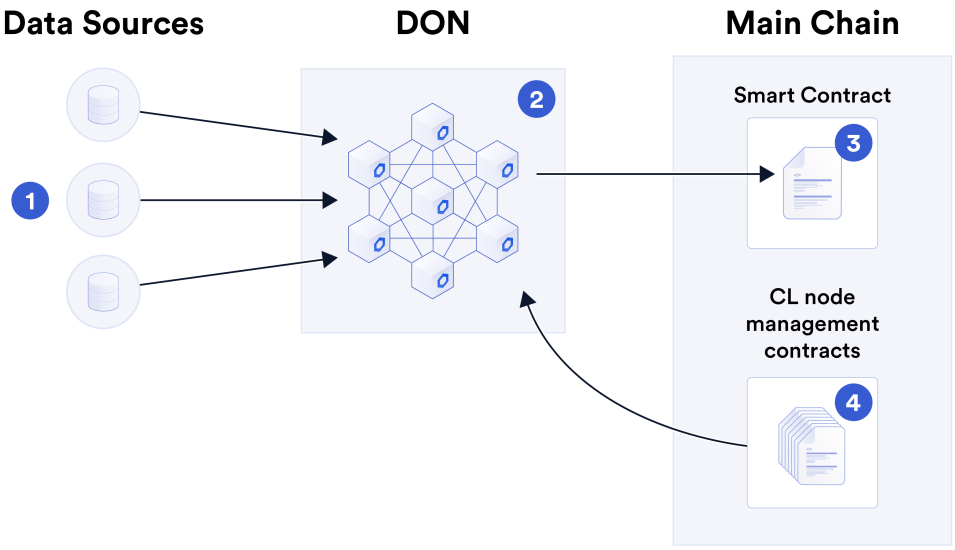

Figure 14: Loci of trust-minimizing mechanisms discussed in this section. ① Data sources provide data to the ② DON, which relays a function of the data to a dependent ③ smart contract. Additionally, the DON or the oracle network includes ④ node management smart contracts on MAINCHAIN for, e.g., compensating nodes, guard rails, and so forth.

source, then the oracle network cannot feasibly tamper with D. Various encouraging efforts to enable such signing of data have emerged, including OpenID Connect, which is designed primarily for user authentication [9], TLS-N, an academic project aiming to extend TLS [191] by repurposing TLS certificates, and TLS Evidence Extensions [63]. While OpenID Connect has seen some adoption, however, TLS Evidence Extensions and TLS-N have yet to see adoption.

Another potential avenue of data-source authentication is to use publishers' own Signed HTTP Exchanges (SXG) [230], which they can cache on content-delivery networks as part of the Accelerated Mobile Pages (AMP) protocol [225]. The Chrome mobile browser displays the content from AMP-cached SXGs as if they were served from their publishers' own network domains instead of the cache-server domain. This branding incentive, coupled with the relative ease of enabling it using services like CloudFlare's Real URL [83] and Google's `amppackager` [124], may lead to widespread adoption of SXGs in cached news content, which would enable a simple, tamper-resistant way for Chainlink oracles to trigger on newsworthy events reported in valid SXGs. While AMP-cached SXGs from news publishers would not be useful for high-tempo applications like reports on trading data, they could be a secure source for custom contracts pertaining to real-world events like extreme weather or election outcomes.

We believe that simple deployment, mature tools, and flexibility will be vital to accelerating data-source signing. Enabling data providers to use Chainlink nodes as an authenticated API front end seems a promising approach. We intend to create an

option for nodes to function in this mode, with or without participation in the network as a full-blown oracle. We refer to this capability as *authenticated data origination* (ADO). By using Chainlink nodes with ADO, data sources will be able to benefit from the experience and tools developed by the Chainlink community in adding digital signing capabilities to their existing suite of off-chain APIs. Should they choose to run their nodes as oracles, they can additionally open up potential new revenue streams under the same model as existing data providers, e.g., Kraken [28], Kaiko [140], and others, that run Chainlink nodes to sell API data on chain.

7.1.1 The Limitations of Authenticated Data Origination

Digital signing by data sources, while it can help strengthen authentication, isn't sufficient *per se* to accomplish all of the natural security or operational goals of an oracle network.

To begin with, a given piece of data D must still be *relayed* in a robust and timely way from a data source to smart contract or other data consumer. That is, even in an ideal setting in which all data is signed using keys pre-programmed into dependent contracts, a *DON* would still be needed to communicate the data reliably from sources to contracts.

Additionally, there are a number of cases in which contracts or other oracle-data consumers want access to authenticated output of various *functions* computed over source data for two main reasons:

- *Confidentiality*: A data source API may provide sensitive or proprietary data that needs to be redacted or sanitized before it is made publicly visible on chain. Any modification to signed data, however, invalidated the signature. Put another way, naïve ADO and data sanitization are incompatible. We show in Example 3 how the two can be reconciled through an enhanced form of ADO.

- *Data source faults:* Both errors and failures can affect data sources, and digital signatures address neither problem. From its inception [98], Chainlink has already included a mechanism to remediate such faults: redundancy. The reports issued by oracle networks typically represent the *combined data* of multiple sources.

We now discuss schemes we are exploring in the ADO setting to enhance the confidentiality of source data and to combine data from multiple sources securely.

7.1.2 Confidentiality

Data sources may not anticipate and make available the full gamut of APIs desired by users. Specifically, users may wish to access *pre-processed* data to help ensure confidentiality. The following example illustrates the problem.

Example 3. *Alice wishes to obtain a decentralized identity (DID) credential stating that she is over 18 years of age (and thus can, for instance, take out a loan). To do so, she needs to prove this fact about her age to a DID credential issuer.*

Alice hopes to use data from her state's Department of Motor Vehicles (DMV) website for the purpose. The DMV has a record of her birthdate and will emit a digitally signed attestation A on it of the following form:

$$A = \{\texttt{Name: Alice}, \texttt{DoB: 02/16/1999}\}.$$

In this example, the attestation A may be sufficient for Alice to prove to the DID credential issuer that she's over 18. But it *needlessly leaks sensitive information*: Alice's exact DoB. Ideally, what Alice would like from the DMV instead is a signature on a simple statement A' that "Alice is over 18 years of age." In other words, she wants the output of a function G on her birthdate X, where (informally), $A' = G(X) = \texttt{True}$ if $\mathsf{CurrentDate} - X \geq 18$ years; otherwise, $G(X) = \texttt{False}$.

To generalize, Alice would like to be able to request from the data source a signed attestation A' of the form:

$$A' = \{\texttt{Name: Alice}, \texttt{Func:}\mathsf{G(X)}, \texttt{Result: True}\},$$

where $\mathsf{G(X)}$ denotes a specification of a function G and its input(s) X. We envision that a user should be able to provide a desired $\mathsf{G(X)}$ as input with her request for a corresponding attestation A'.

Note that the data source's attestation A' must include the specification $\mathsf{G(X)}$ to ensure that A' is correctly interpreted. In the above example, $\mathsf{G(X)}$ defines the meaning of the Boolean value in A' and thus that `True` signifies the subject of the attestation is over 18 years of age.

We refer to flexible queries in which a user can specify $\mathsf{G(X)}$ as *functional queries*. In order to support use cases like that in Example 3, as well as those involving queries directly from contracts, we intend to include support for functional queries involving simple functions G as part of ADO.

7.1.3 Combining Source Data

To reduce on-chain costs, contracts are generally designed to consume *combined* data from multiple sources, as illustrated in the following example.

Example 4 (Medianizing price data)**.** *To provide a price feed, i.e., the value of one asset (e.g., ETH) with respect to another (e.g., USD), an oracle network will generally obtain current prices from a number of sources, such as exchanges. The oracle network typically sends to a dependent contract* SC *the* median *of these values.*

In an environment with data signing, a correctly functioning oracle network obtains from data sources $\mathcal{S} = \{S_1, \ldots, S_{n_S}\}$ a sequence of values $V = \{v_1, v_2, \ldots, v_{n_S}\}$ from n_S sources with accompanying source-specific signatures $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_{n_S}\}$. Upon verifying the signatures, it transmits the price $v = \mathsf{median}(V)$ to SC.

Unfortunately, there is no simple way for an oracle network to transmit the median value v in Example 4 to SC along with a *succinct* proof σ^* that v was *correctly computed* over *signed inputs*.

A naïve approach would be to encode in SC the public keys of all n_S data sources. The oracle network would then relay (V, Σ) and allow SC to compute the median of V. This, however, would result in a proof σ of size $O(n_S)$—i.e., σ^* would not be succinct. It would also incur high gas costs for SC, which would need to verify all signatures in Σ.

Use of SNARKs, in contrast, enables a succinct proof of correctly combined authenticated source values. It may be workable in practice, but imposes fairly high computational costs on the prover, and somewhat high gas costs on chain. Use of Town Crier is also a possibility, but requires the use of TEEs, which does not suit all users' trust models.

A helpful concept in which to frame solutions to the general problem of signing combined data from sources is a cryptographic tool known as *functional signatures* [59, 132]. Briefly, functional signatures allow a signer to delegate signing capability, such that the delegatee can only sign messages in the range of a function F chosen by the signer. We show in Appendix D how this functional constraint can serve to bound the range of report values emitted by a *DON* as a function of the values signed by data sources. We also introduce a new primitive, called a *discretized functional signature*, that includes a relaxed requirement for accuracy, but is potentially much more performant than approaches such as SNARKs.

The problem of combining data sources in a way that includes source authentication of outputs also applies to *data aggregators*, e.g., CoinCap, CoinMarketCap, CoinGecko, CryptoCompare, etc., which obtain data from a multiplicity of exchanges, which they weight based on volumes, using methodologies that they in some cases make public and are in other cases proprietary. An aggregator that wishes to publish a value with source authentication faces the same challenge as a collection of nodes aggregating source data.

7.1.4 Processing Source Data

Sophisticated smart contracts are likely to depend on custom aggregate statistics over primary data sources, such as volatility in recent price history over many assets, or text and photographs from news about pertinent events.

Because computation and bandwidth are relatively cheap in a DON, these statistics—even complex machine-learning models with many inputs—can be processed economically, as long as any output value destined for a blockchain is sufficiently concise.

For computationally intensive jobs where DON participants may have differing views on complex inputs, extra rounds of communication between the DON participants may be required to establish consensus on the inputs before computing the result. As long as the final value is fully determined by the inputs, once input consensus is established each participant can simply compute the value and broadcast it to the other

participants with their partial signature, or send it to an aggregator.

7.2 DON Trust Minimization

We envision two main ways of minimizing the trust placed in components of the DON: *failover clients* and *minority reports*.

7.2.1 Failover Clients

Adversarial models in the cryptography and distributed systems literature typically consider an adversary capable of corrupting (i.e., compromising) a subset of nodes, e.g., fewer than one-third for many BFT protocols. It is commonly observed, however, that if all nodes run identical software, an adversary that identifies a fatal exploit could in principle compromise all nodes more or less simultaneously. This setting is often referred to as a *software monoculture* [47].

Various proposals for automatically diversifying software and software configurations have been put forth to address the problem, e.g., [47, 113]. As noted in [47], however, software diversity is a complex issue and requires careful consideration. Software diversification, for example, can result in *worse* security than a monoculture if it increases a system's attack surface and thus its possible vectors of attack in excess of the security benefits it offers.

We believe that support for robust *failover clients*—i.e., clients to which nodes can switch in the face of a catastrophic event—is an especially attractive form of software diversification. Failover clients do not increase the number of potential vectors of attack, as they are not deployed as mainline software. They offer clear benefits, however, as a second line of defense. We intend to support failover clients in DONs as a key means of reducing their dependence for security on a single client.

Chainlink already has in place a robust system of failover clients. Our approach involves maintaining previous, battle-tested client versions. Today, for example, Chainlink nodes with Off-Chain Reporting (OCR) as their primary client include support for Chainlink's previous FluxMonitor system if needed. Having been in use for some time, FluxMonitor has received security audits and field testing. It provides the same functionality as OCR, just at higher cost—a cost only incurred on an as-needed basis.

7.2.2 Minority Reports

Given a sufficiently large minority set $\mathcal{O}_{minority}$—a fraction of honest nodes that observe malfeasance by the majority—it can be helpful for them to generate a *minority report*. This is a parallel report or flag, relayed to a dependent contract SC on-chain by $\mathcal{O}_{minority}$. SC can make use of this flag according to its own contract-specific policy. For example, for a contract in which safety is more important than liveness or responsiveness, a minority report might cause the contract to request supplementary reports from another DON, or trigger a circuit breaker (see the next section).

Minority reports can play an important role even when the majority is honest, because any report-aggregation scheme, even if it uses functional signatures, must operate in a *threshold* manner, to ensure resilience against oracle or data failure. In other words, it must be possible to produce a valid report based on the inputs of $k_S < n_S$ oracles, for some threshold k_S. This means a corrupted DON has some latitude in manipulating report values by selecting its preferred k_S values among the n_S reported in V by the full set of oracles, *even if all sources are honest*.

For example, suppose that $n_S = 10$ and $k_S = 7$ in a system that uses a functional signature to authenticate computation of median over V for the USD price of ETH. Suppose that five sources report a price of \$500, while the other five report \$1000. Then by medianizing the lowest 7 reports, the DON can output a valid value $v = \$500$, and by medianizing the highest, it can output $v = \$1000$.

By enhancing the DON protocol so that all nodes are aware of which data was available, and which data was used to construct a report, nodes could detect and flag statistically significant tendencies to favor one set of reports over another, and produce a minority report as a result.

7.3 Guard Rails

Our trust model for DONs treats MAINCHAIN as a higher-security, higher-privilege system than DONs. (While this trust model may not always hold true, it is easier to adapt the resulting mechanism to situations where the DON is the higher security platform than *vice versa*.)

A natural trust minimization strategy thus involves the implementation of monitoring and failsafe mechanisms in smart contracts—either in a MAINCHAIN front end for a DON or directly in a dependent contract SC. We refer to these mechanisms as *guard rails*, and enumerate some of the most important here:

- *Circuit breakers:* SC may pause or halt state updates as a function either of characteristics of the state updates themselves (e.g., large variance across sequential reports) or based on other inputs. For example, a circuit breaker might trip in cases where oracle reports vary implausibly over time. A circuit breaker might also be tripped by a minority report. Thus, circuit breakers can prevent DONs from making grossly erroneous reports.

  Circuit breakers can provide time for additional interventions to be considered or exercised. One such intervention is *escape hatches*.

- *Escape hatches:* Under adverse circumstances, as identified by a set of custodians, community token holders, or other bodies of trustees, a contract may invoke an emergency facility sometimes called an *escape hatch* [163]. An escape hatch causes SC to shut down in some manner and/or terminates pending and possibly future transactions. For example, it may return custodied funds to users [17]),

may terminate contract terms [162], or may cancel pending and/or future transactions [173]. Escape hatches can be deployed in any type of contract, not just one that relies on a DON, but they are of interest as a potential buffer against DON malfeasance.

- *Failover:* In systems where SC relies on the DON for essential services, it is possible for SC to provide *failover mechanisms* that ensure service continuation even in the case of DON failure or misbehavior. For example, in the TEF (Section 6), the anchor contract SC_a may provide *dual interfaces* where both on-chain and off-chain execution interfaces are supported for certain critical operations (e.g., withdrawal), or for ordinary transactions, with a suitable delay to prevent front-running of DON transactions. In cases where data sources sign data, users could also furnish reports to SC_a when the DON fails to do so.

Fraud proofs, as proposed for various forms of optimistic rollup (see Section 6.3), are similar in flavor and complementary to the mechanisms we enumerate above. They too provide a form of on-chain monitoring and protection against potential failures in off-chain system components.

7.4 Trust-Minimized Governance

Like all decentralized systems, the Chainlink network requires governance mechanisms to adjust parameters over time, respond to emergencies, and guide its evolution.

Some of these mechanisms currently reside on MAINCHAIN, and may continue to do so even with the deployment of DONs. One example is the payment mechanism for oracle node providers (DON nodes). DON front end contracts on MAINCHAIN contain additional mechanisms, such as guard rails, that may be subject to periodic modification.

We foresee two classes of governance mechanisms: *evolutionary* and *emergency*.

Evolutionary governance: Many modifications to the Chainlink ecosystem are such that their implementation is not a matter of urgency: Performance improvements, feature enhancements, (non-urgent) security upgrades, and so forth. As Chainlink progressively moves toward even more participants in its governance, we expect many or most such changes to be ratified by the community of a specific DON affected by those changes. In the interim, and perhaps ultimately as a parallel mechanism, we believe that a notion of *temporal least privilege* can be a useful means of implementing evolutionary governance. Very simply, the idea is for changes to deploy gradually, ensuring the community an opportunity to respond to them. For example, migration to a new MAINCHAIN contract can be constrained so that the new contract must be deployed at least thirty days before activation.

Emergency governance: Exploitable or exploited vulnerabilities in MAINCHAIN contracts or other forms of liveness or safety failures may require immediate intervention to ensure against catastrophic outcomes. Our intention is to support a multisig intervention mechanism in which, to ensure against malfeasance by any organization, signers will be dispersed across organizations. Ensuring consistent availability of signers and timely access to appropriate chains of command for authorization of emergency changes will clearly require careful operational planning and regular review. These challenges are similar to those involved in testing other cybersecurity incident-response capabilities [134], with a similar need to combat common problems like vigilance decrement [223].

The governance of DONs differs from that of many decentralized systems in its potential degree of heterogeneity. Each DON may have distinct data sources, executables, service-level requirements such as uptime, and users. The Chainlink network's governance mechanisms must be flexible enough to accommodate such variations in operational goals and parameters. We are actively exploring design ideas and plan to publish research on this topic in the future.

7.5 Public-Key Infrastructure

With progressive decentralization will come the need for a robust identification of network participants, including DON nodes. In particular, Chainlink requires a strong Public-Key Infrastructure (PKI). A PKI is a system that binds keys to identities. For example, a PKI undergirds the Internet's system of secure connections (TLS): When you connect to a website via HTTPS (e.g., `https://www.chainlinklabs.com`) and a lock appears in your browser, that means that the public key of the domain owner has been bound to that owner by an authority—specifically, through a digital signature in a so-called *certificate*. A hierarchical system of *certificate authorities* (CAs), whose top-level *root* authorities are hardwired into popular browsers, helps ensure that certificates are issued only to the legitimate owners of domains.

We expect that Chainlink will eventually make use of decentralized name services, initially the Ethereum Name Service (ENS) [22], as the foundation for our PKI. As its name suggests, ENS is analogous to DNS, the Domain Name System that maps (human-readable) domain names to IP addresses on the internet. ENS, however, instead maps human-readable Ethereum names to blockchain addresses. Because ENS operates on the Ethereum blockchain, barring key compromise, tampering with its namespace is in principle as difficult as tampering with the contract administering it and/or the underlying blockchain. (DNS, in contrast, has historically been vulnerable to spoofing, hijacking, and other attacks.)

We have registered `data.eth` with ENS on the Ethereum mainnet, and intend to establish it as a root namespace under which the identities of oracle data services and other Chainlink network entities reside.

Domains in ENS are hierarchical, meaning that each domain may contain references to other names under it. Subdomains in ENS can serve as a way to organize and

delegate trust.

The main role of `data.eth` will be to serve as an on-chain directory service for data feeds. Traditionally, developers and users of oracles have used off-chain sources (e.g., websites like `docs.chain.link` or `data.chain.link`, or social networks such as Twitter) to publish and obtain oracle data feed addresses (such as the ETH-USD price feed). With a highly trustworthy root namespace such as `data.eth`, it is possible instead to establish a mapping of `eth-usd.data.eth` to, e.g., the smart contract address of an on-chain oracle network aggregator for the ETH-USD price feed. This would create a secure path for anyone to refer to the blockchain as the source of truth for that data feed of that price/name pair (ETH-USD). Consequently, such use of ENS realizes two benefits unavailable in off-chain data sources:

- *Strong security:* All changes and updates to the domain are recorded immutably and secured cryptographically, as opposed to text addresses on a website, which enjoy neither of these two security properties.
- *Automated on-chain propagation:* Updates to the underlying address of a data-feed's smart contract can trigger notifications that propagate to dependent smart contracts and can, for example, automatically update dependent contracts with the new addresses.[13]

Namespaces like ENS, however, do not automatically validate *legitimate ownership of asserted names*. Thus, for example, if the namespace includes the entry

$$\langle \text{“}\texttt{Acme Oracle Node Co.}\text{”}, \texttt{addr} \rangle,$$

then a user obtains the assurance that `addr` belongs to the claimant of the name `Acme Oracle Node Co`. Without additional mechanisms around namespace administration, however, she does not obtain assurance that the name belongs to an entity legitimately called Acme Oracle Node Co. in a meaningful real world sense.

Our approach to validation of names, i.e., ensuring their ownership by corresponding, legitimate real-world entities, relies on several components. Today, Chainlink Labs effectively acts as a CA for the Chainlink network. While Chainlink Labs will continue to validate names, our PKI will evolve into a more decentralized model in two ways:

- *Web-of-trust model:* The decentralized counterpart of a hierarchical PKI is often referred to as a *web-of-trust*.[14] Variants have been proposed since the 1990s, e.g., [98], and a number of researchers have observed that blockchains can facilitate use of the idea, e.g., [227] by recording certificates in a globally consistent ledger. We are exploring variants of this model to validate the identities of entities in the Chainlink network in a more decentralized way.

[13] A dependent contract can optionally include a predetermined delay to allow for manual inspection and intervention by dependent-contract administrators.

[14] A term coined by Phil Zimmermann for PGP [238].

- *Linkage to validating data:* Today, a substantial amount of oracle node performance data is visible on-chain, and thus archivally bound to node addresses. Such data may be viewed as enriching an identity in the PKI by providing historical evidence of its (reliable) participation in the network. Additionally, tools for decentralized identity based on DECO and Town Crier [160] enable nodes to accumulate credentials derived from real-world data. As just one example, a node operator can attach a credential to its PKI identity that proves possession of a Dun and Bradstreet rating. These supplementary forms of validation can supplement staking in creating assurance of the security of the network. An oracle node with an established real-world identity may be viewed as having stake in a system deriving from its reputation. (See Section 4.3 and Section 9.6.3.)

A final requirement for the Chainlink PKI is secure *bootstrapping*, i.e., securely publishing the root name for the Chainlink network, currently `data.eth` (analogously to hardwiring of top-level domains in browsers). In other words, how do Chainlink users determine that `data.eth` is indeed the top-level domain associated with the Chainlink project? The solution to this problem for the Chainlink network is multi-pronged and may involve:

- Adding a TXT record [224] to our domain record for `chain.link` that specifies `data.eth` as the root domain for the Chainlink ecosystem. (Chainlink thus implicitly leverages the PKI for internet domains to validate its root ENS domain.)
- Linking to `data.eth` from Chainlink's existing website, e.g., from `https://docs.chain.link`. (Another implicit use of the PKI for internet domains.)
- Making the use of `data.eth` known via various documents, including this whitepaper.
- Posting `data.eth` publicly on our social-media channels, such as Twitter, and the Chainlink blog [18].
- Placing a large quantity of LINK under the control of the same registrant address as `data.eth`.

8 DON Deployment Considerations

While not a part of our core design, there are several important technical considerations in the realization of DONs that deserve treatment here.

8.1 Rollout Approach

This paper lays out an ambitious vision of advanced Chainlink functionality whose realization will require solutions to many challenges along the way. This whitepaper identifies some challenges, but unanticipated ones are sure to arise.

We plan to implement elements of this vision in an incremental fashion over an extended period of time. Our expectation is that DONs will initially launch with support for specific pre-built components built collaboratively by teams within the Chainlink community. The intention is that broader uses of DONs, e.g., the ability to launch arbitrary executables, will see support at a later time.

One reason for such caution is that composition of smart contracts can have complex, unintended, and dangerous side effects, as recent flash-loan-based attacks have for instance shown [127, 189]. Similarly, composition of smart contracts, adapters, and executables will require extreme care.

In our initial deployment of DONs, we plan to include only a pre-built set of templatized executables and adapters. This will enable study of the compositional security of these functionalities using formal methods [46, 170] and other approaches. It will also simplify pricing: Functionality pricing can be established by DON nodes on a per-functionality basis, rather than through generalized metering, an approach adopted in, e.g., [156]. We also expect the Chainlink community to take part in the creation of additional templates, combining various adapters and executables into increasingly useful decentralized services that can be run by hundreds, if not thousands of individual DONs.

Additionally, this approach can help prevent *state bloat*, i.e., the need for DON nodes to retain an unworkable amount of state in working memory. This problem is already arising in permissionless blockchains, motivating approaches such as "stateless clients" (see, e.g., [206]). It can be more acute in higher throughput systems, motivating an approach in which a DON deploys only state-size-optimized executables.

As DONs evolve and mature and include robust guard rails, as discussed in Section 7, cryptoeconomic and reputation-based security mechanisms as discussed in Section 9, and other features that provide a high degree of assurance for DON users, we also expect to develop a framework and tools to facilitate broader launch and use of DONs by the community. Ideally, these tools will enable a collection of node operators to come together as an oracle network and launch their own DONs in a permissionless or self-service manner, meaning that they can do so unilaterally.

8.2 Dynamic DON Membership

The set of nodes running a given DON may change over time. There are two approaches to key management for $\mathsf{sk}_{\mathcal{L}}$ given *dynamic* membership in $\mathcal{O}$.

The first is to update shares of $\mathsf{sk}_{\mathcal{L}}$ held by the nodes upon changes in membership, while keeping $\mathsf{pk}_{\mathcal{L}}$ unchanged. This approach, explored in [41, 161, 198], has the merit of not requiring that relying parties update $\mathsf{pk}_{\mathcal{L}}$.

The classical technique of *share resharing*, introduced in [122], provides a simple and efficient way of realizing such share updates. It enables a secret to be transferred between one set of nodes $\mathcal{O}^{(1)}$ and a second, possibly intersecting one $\mathcal{O}^{(2)}$. In this approach, each node $\mathcal{O}_i^{(1)}$ performs a $(k^{(2)}, n^{(2)})$ secret sharing of its secret share across nodes in $\mathcal{O}^{(2)}$ for $n^{(2)} = |\mathcal{O}^{(2)}|$ and desired (possibly new) threshold $k^{(2)}$. Various verifiable secret sharing (VSS) schemes [108] can provide security against an adversary that actively corrupts nodes, i.e., introduces malicious behavior into the protocol. Techniques in [161] aim to do so while reducing communication complexity and providing resilience against failures in cryptographic hardness assumptions.

A second approach is to update the ledger key $\mathsf{pk}_{\mathcal{L}}$. This has the benefit of *forward security*: Compromise of old shares of $\mathsf{pk}_{\mathcal{L}}$ (i.e., former committee nodes) would not result in compromise of the current key. Updates to $\mathsf{pk}_{\mathcal{L}}$, however, carry two drawbacks: (1) Data encrypted under $\mathsf{pk}_{\mathcal{L}}$ needs to be re-encrypted during a key refresh and (2) Key updates need to be propagated to relying parties.

We intend to explore both approaches, as well as hybridizations of the two.

8.3 DON Accountability

As with existing Chainlink oracle networks, DONs will include mechanisms for *accountability*, i.e., recording, monitoring, and enforcing correct node behavior. DONs will have much more substantial data capacity than many existing permissionless blockchains, particularly given their ability to connect to external decentralized storage. Consequently, they will be able to record nodes' performance history in detail, allowing for more fine-grained accountability mechanisms. For example, off-chain computation of asset prices may involve inputs that are discarded before a median result is sent on chain. In a DON, these intermediate results could be recorded. Misbehavior or performance lapses by individual nodes in a DON can thus be remedied or penalized on the DON in a fine-grained way. We have additionally discussed approaches to building guard rails in Section 7.3 that address the contract-specific impact of systemic failures.

It is also important, however, to have failsafe mechanisms for DONs themselves, i.e., protections against systemic, potentially catastrophic DON failures, specifically *forking / equivocation* and service-level agreement (SLA) failures, as we now explain.

Forking / equivocation: Given sufficiently many faulty nodes, a DON can *fork* or *equivocate*, producing two distinct, inconsistent blocks or sequences of blocks in $\mathcal{L}$. Because a DON digitally signs the contents of $\mathcal{L}$, however, it is possible to leverage a main chain MAINCHAIN to prevent and/or penalize equivocation.

The DON can periodically checkpoint state from $\mathcal{L}$ in an audit contract on MAINCHAIN. If its future state deviates from a checkpointed state, a user / auditor can present proof of this misbehavior to the audit contract. Such proof can be used to generate an alert or penalize DON nodes via slashing in the contract. This latter approach introduces an incentive design problem similar to that for specific oracle feeds, and can build on our work outlined in Section 9.

Enforcing service-level agreements: While DONs are not necessarily meant to run indefinitely, it is important that they adhere to service level agreements (SLAs) with their users. Basic SLA enforcement is possible on a main chain. For example, DON nodes might commit to maintaining the DON until a certain date, or to providing advance notice of service termination (e.g., three months' notice). A contract on MAINCHAIN can provide basic cryptoeconomic SLA enforcement.

For example, the SLA contract can slash DON-deposited funds if checkpoints are not provided at required intervals. A user can deposit funds and challenge the DON to prove that a checkpoint correctly represents a sequence of valid blocks (in a manner analogous to, e.g. [141]). Of course, block production does not equate with transaction processing, but the SLA contract can also serve to enforce the latter. For example, in the legacy-compatible version of FSS in which transactions are fetched from the mempool (see Section 5.2), transactions are eventually mined and placed on chain. A user can prove DON malfeasance by furnishing the SLA contract with a transaction that was mined but wasn't transmitted by the DON for processing by the target contract within the appropriate interval of time.[15]

It is also possible to prove the existence of and penalize more fine-grained SLA failures, including errors in computation using executables (via, e.g., the mechanisms for proving correct off-chain state transactions outlined in Section 6.3) or failure to run executables based on initiators visible on a DON, failure to relay data on the DON to MAINCHAIN in a timely way, and so forth.

9 Economics and Cryptoeconomics

For the Chainlink network to achieve strong security within a decentralized trust model, it is essential that nodes collectively exhibit correct behavior, meaning that they adhere a majority of the time exactly to DON protocols. In this section, we discuss approaches to helping enforce such behavior by means of *economic incentives*, a.k.a. *cryptoeconomic incentives*. These incentives fall into two categories: *explicit* and *implicit*, realized respectively through *staking* and *future fee opportunity (FFO)*.

Staking: Staking in Chainlink, as in other blockchain systems, involves network participants, i.e., oracle nodes, depositing locked funds in the form of LINK tokens. These funds, which we also refer to as *stake* or *explicit stake* are an *explicit incentive*. They are subject to forfeiture upon node failure or malfeasance. In the blockchain context, this procedure is often called *slashing*.

Staking by oracle nodes in Chainlink, however, differs fundamentally from staking by validators in permissionless blockchains. Validators can misbehave by equivocating or adversarially ordering transactions. The underlying consensus protocol in a

[15] As users can replace transactions in the mempool, care is required to ensure a correct correspondence between the mined and DON-submitted transactions.

permissionless blockchain, though, uses hard-and-fast block-validation rules and cryptographic primitives to prevent validators from generating invalid blocks. In contrast, programmatic protections *cannot* prevent a cheating oracle network from generating invalid reports. The reason is a key difference between the two types of system: transaction validation in blockchains is a property of *internal consistency*, while the correctness of oracle reports on a blockchain is a property of *external, i.e., off-chain data*.

We have designed a preliminary staking mechanism for the Chainlink network based on an interactive protocol among oracle nodes that may make use of external data. This mechanism creates financial incentives for correct behavior using explicit rewards and penalties (slashing). As the mechanism is economic, it is designed to prevent node corruption by an adversary that uses financial resources to corrupt nodes by means of *bribery*. (Such an adversary is very general, and extends, e.g., to nodes cooperating to extract value from their collective misbehavior.)

The Chainlink staking mechanism we have designed has some powerful and novel features.[16] The main such feature is *super-linear staking impact* (specifically, *quadratic*). An adversary must have resources *considerably in excess of nodes' deposited funds* in order to subvert the mechanism. Our staking mechanism additionally provides protection against a stronger adversary than previously considered in similar systems, namely an adversary that can create bribes conditioning on nodes' future behavior. Additionally, we discuss how Chainlink tools such as DECO can help strengthen our staking mechanism by facilitating correct adjudication in the case of faulty node behavior.

Future fee opportunity (FFO): Permissionless blockchains—of both the PoW and PoS variety—today rely critically on what we call *implicit incentives*. These are economic incentives for honest behavior that derive not from explicit rewards, but from platform participation itself. For example, the Bitcoin miner community is incentivized against mounting a 51% attack by the risk of undermining confidence in Bitcoin, depressing its value, and consequently eroding the value of their collective capital investments in mining infrastructure [150].

The Chainlink network benefits from a similar implicit incentive that we refer to as *future fee opportunity* (FFO). Oracle nodes with strong performance histories or reputations attract fees from users. Misbehavior by an oracle node jeopardizes future fee payments and thus penalizes the node with an opportunity cost in terms of potential revenue earned through participation in the network. By analogy with explicit stake, FFO may be viewed as a form of *implicit stake*, an incentive for honest behavior that derives from the shared benefit of maintaining confidence in the platform on which node operators' business depends, i.e., the positive performance and reputation of the network. This incentive is inherent in but not explicitly expressed in Chainlink network protocols. In Bitcoin, maintaining the value of mining operations as mentioned above

[16] The staking mechanism we describe here currently aims only to enforce delivery of correct reports by oracle networks. We expect in future work to extend it to ensure correct execution of the many other functionalities DONs will provide.

may similarly be viewed as a form of implicit stake.

We emphasize that FFO *already exists in Chainlink and helps secure the network today*. Our main contribution in the further development of Chainlink will be a principled, empirically driven approach to evaluating implicit incentives such as FFO through what we call the *Implicit-Incentive Framework* (IIF). To estimate quantities such as the future fee opportunity of nodes, the IIF will draw continuously on the comprehensive performance and payment data amassed by the Chainlink network. Such estimates will enable IIF-based parameterization of staking systems that reflects node incentives with greater accuracy than current heuristic and/or static models.

To summarize, then, the two main economic incentives for correct oracle node behavior in the developing Chainlink network will be:

- *Staking (deposited stake)* } Explicit incentive
- *Future fee opportunity (FFO)* } Implicit incentive

These two forms of incentive are complementary. Oracle nodes can simultaneously participate in the Chainlink staking protocol, enjoy an ongoing revenue stream from users, and collectively benefit from their continued good behavior. Thus both incentives contribute to the cryptoeconomic security provided by an oracle network. Additionally, the two incentives can reinforce and/or be traded off against one another. For example, a new oracle operator without a performance history and revenue stream can stake a large quantity of LINK as a guarantee of honest behavior, thereby attracting users and fees. Conversely, an established oracle operator with a long, relatively fault-free performance history can charge substantial fees from a large user base and thus rely more heavily on its FFO as a form of implicit incentive.

In general, the approach we consider here aims for a given amount of oracle-network resource to create the greatest possible economic incentives in Chainlink for *rational* agents—i.e., nodes maximizing their financial utility—to behave honestly. Put another way, the goal is to maximize the financial resources required for an adversary to attack the network successfully. By formulating a staking protocol with mathematically well defined economic security and also using the IIF, we aim to measure the strength of Chainlink's incentives as accurately as possible. The creators of relying contracts will then be able to determine with strong confidence whether an oracle network meets their required levels of cryptoeconomic security.

The virtuous cycle of economic security: The incentives we discuss in this section, staking and FFO, have an impact beyond their reinforcement of the security of DONs. They promise to induce what we call a *virtuous cycle of economic security*. Super-linear staking impact (and other economies of scale) result in lower operational cost as a DON's security grows. Lower cost attracts additional users to the DON,

boosting fee payments. A rise in fee payments continues to incentivize growth of the network, which perpetuates the virtuous cycle.

We believe that the virtuous cycle of economic security is just one example of an economy of scale and network effect among others that we discuss later in this section.

Section organization: Staking presents notable technical and conceptual challenges for which we have designed a mechanism with novel features. Staking will therefore be our main focus in this section.

We give an overview of the staking approach we introduce in this paper in Section 9.1, followed by detailed discussion in Sections 9.2 to 9.5. We present the IFF in Section 9.6. We present a summary view of Chainlink network incentives in Section 9.7.

In Section 9.8, we discuss the virtuous cycle of economic security our proposed staking approach can bring to oracle networks. Finally, we briefly describe other potential effects propelling growth of the Chainlink network in Section 9.9.

9.1 Staking Overview

The staking mechanism design we introduce here, as noted above, involves an interactive protocol among oracle nodes allowing for resolution of inconsistencies in the reporting of external data. Staking aims to ensure honest behavior from rational oracle nodes. We can therefore model an adversary attacking a staking protocol as a *briber*: The adversary's strategy is to corrupt oracle nodes using financial incentives. The adversary may derive financial resources prospectively from successfully tampering with an oracle report, e.g., offer to share the resulting profit with corrupted nodes.

We aim in our staking mechanism design simultaneously at two ambitious goals:

1. *Resisting a powerful adversary:* The staking mechanism is designed to protect oracle networks against a broad class of adversaries that are capable of complex, conditional bribing strategies, including *prospective* bribery, which offers bribes to oracles whose identities are determined after the fact (e.g., offers bribes to oracles randomly selected for high-priority alerting). While other oracle designs have considered a narrow set of attacks without the full capabilities of a realistic adversary, to the best of our knowledge the adversarial mechanism we introduce here is the first to explicitly address a broad set of bribing strategies and show resistance in this model. Our model assumes that nodes besides the attacker are economically rational (as opposed to honest), and we assume the existence of a source of truth that is prohibitively expensive for typical usage but is available in case of disagreement (discussed further below).

2. *Achieving super-linear staking impact:*

   Our aim is to ensure that an oracle network composed of rational agents reports truthfully even in the presence of an attacker with a budget that is *super-linear*

in the total stake deposited by the entire network. In existing staking systems, if each of n nodes stakes $\$d$, an attacker can issue a credible bribe which requests that nodes behave dishonestly in exchange for a payment of slightly more than $\$d$ to each node, using a total budget of about $\$dn$. This is already a high bar as the attacker has to have a liquid budget on the order of the combined deposits of all stakers in the network. Our goal is a still stronger degree of economic security than this already substantial hurdle. We aim to design the first staking system that can achieve security for a general attacker with a budget *super-linear* in n.

While practical considerations may achieve a lower impact, as we discuss below, our preliminary design achieves an adversarial budget requirement greater than $\$dn^2/2$, i.e., scaling quadratic in n, rendering bribery largely impractical even when nodes stake only moderate amounts.

Reaching these two goals requires an innovative combination of incentive design and cryptography.

Key ideas: Our staking approach hinges on an idea we call *watchdog priority*.

A report generated by a Chainlink oracle network and sent to a relying contract (e.g., on an asset price) is aggregated from individual reports contributed by participating nodes (e.g., by taking the median). Typically a service-level agreement (SLA) specifies acceptable bounds of deviation for reports, i.e., how far a node's report can deviate from the aggregate report and how far the aggregate should be permitted to deviate from the true value to be considered *correct*.

In our staking system, for a given reporting round, each oracle node can act as a *watchdog* to raise an *alert* if it believes the aggregate report is incorrect. In each reporting round, each oracle node is assigned a public *priority* that determines the order in which its alert (if any) will be processed. Our mechanism aims at reward *concentration*, meaning that the highest-priority watchdog to raise an alert earns the entire reward yielded by confiscating the deposits of faulty nodes.

Our staking system designs involve two *tiers*: the first, default tier, and the second, backstop tier. The first tier is the oracle network itself, a set of n nodes. (For simplicity, we assume n is odd.) If a majority of nodes report incorrect values, a watchdog in the first tier is strongly incentivized to raise an alert. If an alert is raised, the reporting decision of the network is then escalated to a second tier—a high-cost, maximum-reliability system that can be user-specified in the network service-level agreement. This could be a system which, for example, is composed only of nodes with strong historical reliability scores, or one that has an order of magnitude more oracles than the first tier. Additionally, as discussed in Section 9.4.3, DECO or Town Crier can serve as powerful tools to help ensure efficient and conclusive adjudication in the second tier. For simplicity we thus assume that this second-tier system arrives at a correct report value.

While it might seem attractive just to rely on the second tier to generate all reports, the benefit of our design is that it consistently achieves the security properties of the

second-tier system while only paying the operating cost, in the typical case, of the first-tier system.

Watchdog priority results in super-linear staking impact in the following way: if the first-tier oracle network outputs an incorrect result and a number of watchdog nodes alert, the staking incentive mechanism rewards the highest-priority watchdog with more than $\$dn/2$ drawn from the deposits of the (majority) misbehaving nodes. The total reward is thus concentrated in the hands of this single watchdog, which therefore determines the minimum that an adversary must promise a potential watchdog to incentivize it *not* to alert. Since our mechanism ensures that every oracle gets the chance to act as watchdog if the higher-priority watchdogs have accepted their bribes (and chosen not to alert), the adversary must therefore offer a bribe of more than $\$dn/2$ to *every* node to prevent any alert being raised. Since there are n nodes, the adversary's requisite budget for a successful bribe amounts to more than $\$dn^2/2$, which is quadratic in the number n of nodes in the network.

9.2 Background

Our approach to staking draws on research in the fields of *game theory* and *mechanism design* (MD) (for a textbook reference, see [177]). Game theory is the mathematically formalized study of strategic interaction. In this context, a *game* is a model of such an interaction, typically in the real world, that codifies sets of actions available to participants in the game, known as *players*. A game also specifies the *payoffs* obtained by the individual players—rewards that depend on a player's chosen actions and the actions of the other players. Perhaps the best known example of a game studied in game theory is the Prisoners' Dilemma [178]. Game theorists generally aim to understand the *equilibrium* or *equilibria* (if any) represented in a given game. An equilibrium is a set of strategies (one for each player) such that no one player can obtain a higher payoff by unilaterally deviating from its strategy.

Mechanism design, meanwhile, is the science of designing incentives such that the equilibrium of an interaction (and its associated game) has some desirable property. MD may be viewed as the inverse of *game theory*: The canonical question in game theory is, "given the incentives and model, what will the equilibrium be?" In MD, the question is instead, "what incentives will result in a game with a desirable equilibrium?" A typical goal of a mechanism designer is to create an 'incentive compatible' mechanism, meaning that participants in the mechanism (e.g., an auction or other information elicitation system [228]) are incentivized to report the truth on some matter (e.g., how much they value a particular item). The Vickrey (second-price) auction is perhaps the best known incentive compatible mechanism, in which participants submit sealed bids for an item and the highest bidder wins the item but pays the second-highest price [214]. *Cryptoeconomics* is a domain-specific form of MD that leverages cryptographic techniques to create desirable equilibria within decentralized systems.

Bribery and collusion create significant challenges throughout the field of MD. Almost all mechanisms break in the presence of collusion, defined as side contracts be-

tween the parties participating in a mechanism [125, 130]. Bribery, in which an external party introduces novel incentives into the game, presents an even tougher problem than does collusion; collusion may be viewed as a special case of bribery among game participants.

Blockchain systems can often be conceptualized as games with monetary (cryptocurrency-based) payoffs. A simple example is Proof-of-Work mining: miners have an action space in which they can choose the hashrate with which to mine for blocks. The payoff of mining is a guaranteed negative reward (cost of electricity and equipment) plus a stochastic positive reward (mining subsidy) that depends on the number of other active miners [106, 172] and transaction fees. Crowdsourced oracles like SchellingCoin [68] are another example: the action space is the set of possible reports an oracle may send, while the payoff is the reward specified by the oracle mechanism, e.g., payment might depend on how close an oracle's report is to the median of the other reports [26, 68, 119, 185].

Blockchain games offer ripe opportunities for collusion and bribery attacks; indeed, smart contracts can even facilitate such attacks [96, 165]. Perhaps the best known bribery attack on crowdsourced oracles is the *p-plus-epsilon* attack [67]. This attack arises in the context of a SchellingCoin-like mechanism in which players submit boolean-valued reports (i.e., `false` or `true`) and are rewarded with p if they agree with the majority submission. In a p-plus-epsilon attack, the attacker credibly promises to, e.g., pay users $\$p + \epsilon$ for voting `false` if and only if the majority submission is `true`. The result is an equilibrium, in which all players are incentivized to report `false` irrespective of what other players do; consequently, the briber can induce the nodes through its promised bribe to report `false` without actually paying the bribe (!).

Exploration of other briber strategies in the context of oracles, however—and particularly oracles that are not crowdsourced—has been limited to fairly weak adversarial models. For example, in the PoW setting, researchers have studied *outcome-contingent* bribes, i.e., bribes paid only if a target message is successfully censored and does not appear in a block, irrespective of an individual miner's action [96, 165]. In the case of oracles, however, other than the p-plus-epsilon attack, we are aware only of work in a strictly limited model of bribery in which a briber sends a bribe conditioned on an individual player's action, not on the resulting outcome.

Here we sketch designs of information-elicitation mechanisms that remain incentive compatible even in a strong adversarial model, as described in the next subsection.

9.3 Modeling Assumptions

In this subsection, we explain how we model the behavior and capabilities of players in our system, specifically first-tier oracle nodes, nodes in the second-tier (adjudication) layer, and adversaries.

9.3.1 First-Tier Incentive Model: Rational Actors

Many blockchain systems rely for security on the assumption of some number of *honest* participating nodes. Nodes are defined to be honest if they follow the protocol even when it is not in their financial interest to do so. Proof-of-Work systems typically require the majority of hash power to be honest, Proof-of-Stake systems typically require 2/3 or more of all participating stake to be honest, and even layer-2 systems like Arbitrum [141] require at least a single honest participant.

In modeling for our staking mechanism, we make a much weaker assumption. (To be clear, weaker assumptions mean stronger security properties and are therefore preferable.) We assume that the adversary has corrupted, i.e., controls, some (minority) fraction of first-tier oracle nodes. We model the remaining nodes not as honest agents, but as *rational expected-utility maximizers*. These nodes act entirely according to self-interested financial incentives, choosing actions that result in an expected financial gain. For example, if a node is offered a bribe larger than the reward resulting from honest behavior, it will accept the bribe.

Note on adversarial nodes: In accordance with the trust modeling common for decentralized systems, we assume that all nodes are rational, i.e., seeking to maximize net revenue, rather than controlled by a malicious adversary. Our claims, however—specifically super-linear or quadratic staking impact—hold asymptotically provided that the set of adversarially controlled nodes is at most $(1/2 - c)n$, for some positive constant c.

9.3.2 Second-Tier Adjudication Model: Correctness by Assumption

Recall that a critical feature of our staking mechanism that helps achieve security against rational nodes is its second-tier system.

In our proposed staking mechanism, any oracle may raise an alert indicating that it believes the output of the mechanism is incorrect. An alert results in a high-trust second-tier system activating and reporting the correct result. Thus, a key modeling requirement for our approach is *correct adjudication*, i.e., correct reporting by the second-tier system.

Our staking model assumes a second-tier system that acts as an incorruptible, maximally reliable source of truth. Such a system is likely to be *expensive and slow*, and thus inappropriate for use for the typical case. In the equilibrium case, however, i.e., when the first-tier system functions correctly, the second-tier system will not be invoked. Instead, its existence boosts the security of the whole oracle system by providing a high-assurance backstop.

The use of a high-trust, high-cost adjudication layer resembles the appeals process at the heart of most judicial systems. It is also already common in the design of oracle systems, e.g., [119, 185]. We briefly discuss approaches to realization of the second tier in our mechanism in Section 9.4.3.

Our staking protocol uses the assumed correct adjudication of the second-tier system as a *credible threat* to enforce correct reporting by oracle nodes. The protocol confiscates part or all of the stake of oracle nodes that generate reports identified by the second-tier system as incorrect. Oracle nodes are thus deterred from misbehaving by the resulting financial penalty. This approach is similar in flavor to that used in optimistic rollups, e.g., [141, 10].

9.3.3 Adversarial Model

Our staking mechanism is designed to elicit truthful information while achieving security against a broad, well-defined class of adversaries. It improves upon prior works, which either omit an explicit adversarial model or focus on narrow sub-classes of adversaries, e.g., the p-plus-epsilon adversary discussed above. Our goal is to design a staking mechanism with formally proven security against the full spectrum of adversaries likely to be encountered in practice.

We model our adversary as having a fixed (parameterizable) budget, denoted by $\$B$. The adversary can communicate individually and confidentially with each oracle in the network, and can secretly offer any individual oracle guaranteed payment of a bribe contingent on publicly observable outcomes of the mechanism. Outcomes determining bribes can include, for example, the value reported by the oracle, any public messages sent by any oracle to the mechanism (e.g., an alert), the values reported by other oracles, and the value output by the mechanism.

No mechanism can secure against an attacker with unlimited capabilities. We therefore consider some behaviors as unrealistic or out-of-scope. We assume our attacker cannot break standard cryptographic primitives, and, as noted above, has a fixed (if potentially large) budget $\$B$. We further assume that the adversary does not control communication in the oracle network, specifically that it cannot substantially delay traffic between first-tier and/or second-tier nodes. (Whether the adversary can observe such communication depends on the particular mechanism, as we explain below.)

Informally, however, as noted above, we assume that the adversary can: (1) *Corrupt* a fraction of oracle nodes ($(1/2 - c)$-fraction for some constant c), i.e., fully control them, and (2) *Offer bribes* to any desired nodes, with guaranteed payment contingent on outcomes specified by the adversary, as described above.

While we don't offer a formal model or complete taxonomy of the adversary's full range of bribing capabilities in this whitepaper, here are examples of the kinds of bribers encompassed by our model. For simplicity, we assume that oracles emit Boolean reports whose correct value (w.l.o.g.) is `true`, and that a *final outcome* is computed as an aggregate of these reports to be used by a consuming smart contract. The briber's aim is for the final outcome to be incorrect, i.e., `false`.

- *Unconditional briber:* Briber offers bribe $\$b$ to any oracle that reports `false`.
- *Probabilistic briber:* Briber offers bribe $\$b$ with some probability q to any oracle that reports `false`.

- *`false`-outcome conditioned briber:* Briber offers bribe $\$b$ to any oracle that reports `false` provided that the final outcome is `false`.
- *No-alert-conditioned briber:* Briber offers bribe $\$b$ to any oracle that reports `false` as long as no alert is raised.
- *p-plus-epsilon Briber:* Briber offers bribe $\$b$ to any oracle that reports `false` as long as the majority of oracles do not report `false`.
- *Prospective briber:* Briber offers bribe $\$b$ in advance to whichever oracle is selected for a randomized role and reports `false`. In our proposed staking protocol, all nodes act as potential watchdogs, and we are able to show that randomization of watchdog priorities does not lend itself to prospective bribery. Many proof-of-work, proof-of-stake, and permissioned systems *are* susceptible to prospective bribery, however, which shows the importance of considering it in our adversarial model and ensuring that our staking protocols are resilient to it. See Appendix E for more details.

9.3.4 How Much Cryptoeconomic Security Is Enough?

A rational adversary will only spend money to attack a system if it can obtain a profit larger than its expenditure. Thus for our adversarial model and proposed staking mechanism, $\$B$ may be viewed as a measure of the potential profit an adversary is able to extract from relying smart contracts by corrupting an oracle network and causing it to generate an incorrect report or set of reports. In deciding whether an oracle network offers a sufficient degree of cryptoeconomic security for their purposes, a user should assess the network from this perspective.

For plausible adversaries in practical settings, we expect that $\$B$ will generally be substantially smaller than the total assets in relying smart contracts. In most cases, it is infeasible for an adversary to extract these assets in their totality.

9.4 Staking Mechanism: Sketch

Here we present the main ideas and general structure of the staking mechanism we are currently considering. For ease of presentation, we describe a simple but slow (multi-round) protocol in this subsection. We note, however, that this scheme is quite practical. Given the economic assurances provided by the mechanism, i.e., the penalization of and consequent incentive against faulty nodes, many users may be willing to accept reports *optimistically*. In other words, such users may accept reports prior to potential adjudication by the second tier.

Users unwilling to accept reports optimistically can choose to wait until the protocol execution terminates, i.e., until any potential escalation to the second tier occurs. This, however, can substantially slow the confirmation time for reports. We therefore briefly

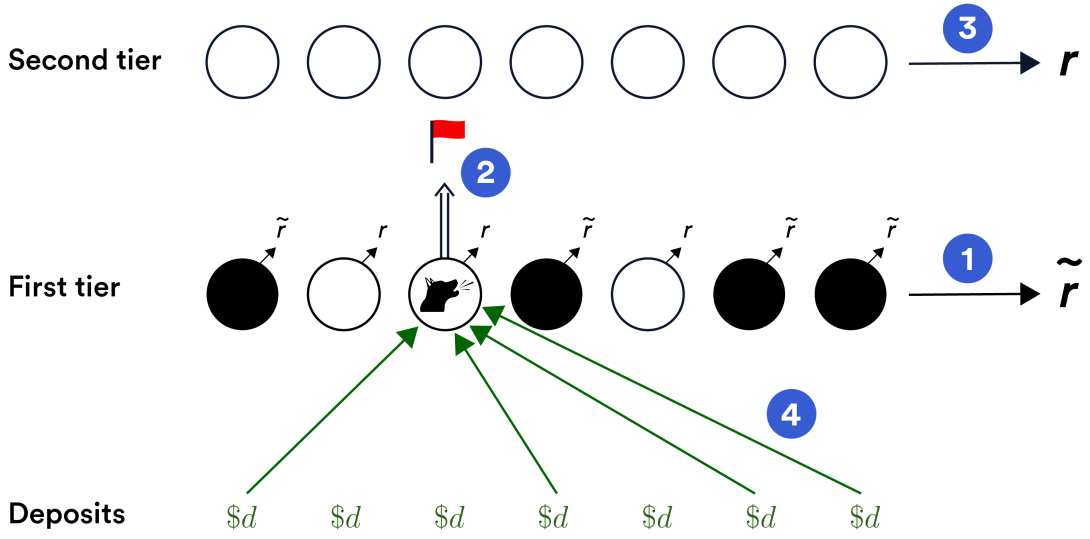

Figure 15: Schematic of staking scheme with alerting. In this example, ① a majority of nodes are corrupted / bribed and emit an incorrect value $\tilde{r}$, rather than the correct report value r. The watchdog node ② sends an alert to the second-tier committee, which ③ determines and emits the correct report value r, resulting in corrupted nodes forfeiting their deposits—each $\$d$ to the watchdog node ④.

outline some optimizations that result in a faster (single-round) if somewhat more complex design in Section 9.5.

Recall that the first tier in our staking mechanism consists of the basic oracle network itself.

The main structure of our mechanism, as described above, is that in each round, each node can act as a "watchdog" with some priority, and it thus has the ability to raise an alert if the mechanism arrives at an incorrect output $\tilde{r}$, rather than a correct one r. This alert causes second-tier resolution, which we assume arrives at a correct report. Nodes with incorrect reports are punished, in the sense that their stakes are slashed and awarded to watchdogs. This basic structure is common in oracle systems, as in, e.g., [119, 185].

The key innovation in our design, mentioned briefly above, is that every node is assigned a distinct *priority* in the ordering of potential watchdogs. That is, watchdogs are given opportunities to alert in priority sequence. Recall that if a node has the highest priority to raise an alert, it receives the slashed deposit $\$d$ of every misbehaving node, for a total of more than $\$dn/2 = \$d \times n/2$, as an incorrect report implies a majority of bad nodes. Consequently, the adversary must pay at least this reward to bribe an arbitrary node. Thus, to bribe a majority of nodes, the adversary *must pay a large bribe to a majority of nodes*, namely, strictly more than $\$dn^2/2$.

We show schematically how alerting and watchdog escalation works in Fig. 15.

9.4.1 Further Mechanism Details

The bribery-resistant system we now describe in further detail is a simplified sketch of the two-tiered construction we intend to build. Most of our focus will be on describing the first-tier network (henceforth simply "network" where clear from context) along with its incentive mechanism and the procedure for escalation to the second tier.

Consider a Chainlink network composed of n oracle nodes that are responsible for regularly (e.g., once a minute) reporting a boolean value (e.g., whether the market capitalization of BTC exceeds that of ETH). As part of the staking mechanism, nodes must provide two deposits: a deposit $\$d$ subject to slashing in the event of disagreement with the majority and a watchdog deposit $\$d_w$ subject to slashing in the event of a faulty escalation. We assume that the nodes cannot copy the submissions of other nodes, e.g., through a commit-reveal scheme as discussed in Section 5.3. In each round, nodes first commit to their report, and once all nodes have committed (or a timeout has expired), nodes reveal their reports.

For each report to be generated, every node is also given a watchdog priority between 1 and n chosen at random, with 1 being top priority. This priority enables the concentration of reward in the hands of one watchdog. After all reports are public, an alerting phase ensues. Over a sequence of n (synchronous) rounds, the node with priority i has the opportunity to alert in round i.

Let us consider the possible outcomes for the mechanism after nodes have revealed their reports. Again assuming a binary report, suppose the correct value is `true` and the incorrect one is `false`. Suppose also that the first-tier mechanism outputs the majority value output by nodes as the final report r.

There are three possible outcomes in the mechanism:

- *Complete agreement:* In the best case, nodes are in complete agreement: all nodes are available and have provided a timely report of the same value r (either `true` or `false`). In this case, the network need only forward r to relying contracts and reward each node with a fixed per-round payment $\$p$, which is much smaller than $\$d$.

- *Partial agreement:* It is possible that some nodes are offline or there is disagreement about which value is correct, but most nodes report `true` and only a minority reports `false`. This case is also straightforward. The majority value (`true`) is computed, resulting in a correct report r. All nodes that reported r are rewarded with $\$p$ while the oracles that reported incorrectly have their deposits slashed modestly, e.g., by $\$10p$.

- *Alert:* In the event that a watchdog believes the output of the network is incorrect, it publicly triggers an alert, escalating the mechanism to the second-tier network. There are then two possible results:

  – *Correct alert:* If the second-tier network confirms that the output of the

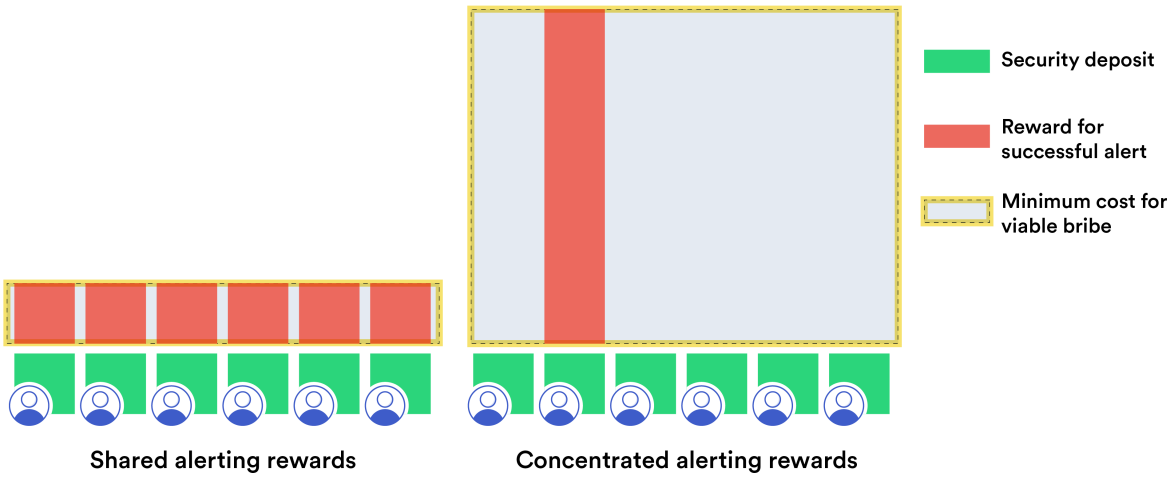

Figure 16: Amplifying briber's cost through concentrated alerting rewards. A bribing adversary must bribe each node with more than the reward it stands to gain by alerting (shown as a red bar). If alerting rewards are shared, then this reward may be relatively small. Concentrated alerting rewards increase the reward that any single node may obtain (tall red bar). Consequently the total payout by the adversary for a viable bribe (gray regions) is much larger with concentrated than shared alerting rewards.

first-tier network was incorrect, the alerting watchdog node receives a reward consisting of all slashed deposits, and thus more than $\$dn/2$.

– *Faulty alert:* If the second-tier and first-tier oracles agree, the escalation is deemed faulty and the alerting node loses its $\$d_w$ deposit.

In the case of optimistic acceptance of reports, watchdog alerts do not cause any change in execution of relying contracts. For contracts designed to await potential arbitration by the second-tier committee, watchdog alerts delay but do not freeze contract execution. It is also possible for contracts to designate a failover DON for periods of adjudication.

9.4.2 Quadratic Staking Impact

The ability for every node to act as a watchdog, combined with strict node priority ensuring concentrated rewards, enables the mechanism to achieve quadratic staking impact for each kind of bribing attacker described in Section 9.3.3. Recall that this means specifically in our setting that, for a network with n nodes each with deposit $\$d$, a successful briber (of any of the kinds above) must have a budget of bigger than $\$dn^2/2$.

To be precise, the briber must corrupt at least $(n+1)/2$ nodes, since the briber must corrupt a majority of n nodes (for odd n, by assumption). Thus, a watchdog stands to earn a reward of $\$d(n+1)/2$. The briber consequently must pay this amount to every

node to ensure that none acts as a watchdog. We are working to show formally that if the briber has a budget of at most $\$d(n^2+n)/2$, then the *subgame perfect equilibrium* of the game between the bribers and the oracles—in other words, the equilibrium at any point during the play of the game—is for the briber not to issue the bribe and for each oracle to report its true values honestly.

We have explained above how it is possible that a successful briber could require a budget significantly larger than that of the sum of the node deposits. To illustrate this intuitive result, Fig. 16 shows the impact of concentrated alert rewards graphically. As we see there, if the reward for watchdog alerting—namely the deposits of bribed nodes reporting `false`)—were split among all potential alerting, the total amount that any individual alerting node could expect would be relatively small, on the order of $\$d$. A briber, knowing that a payout of larger than $\$d$ was improbable, could use a `false`-outcome conditional bribe to bribe each of n nodes with slightly more than $\$d+\epsilon$.

Counterintuitively, Fig. 16 shows that a system that distributes a reward broadly among nodes signaling an alert is far weaker than one that concentrates the reward in the hands of a single watchdog.

Example parameters: Consider a (first-tier) network with $n = 100$ nodes, each depositing $\$d = \$20K$. This network would have a total of $\$2M$ deposited but would be protected against a briber with budget $\$100M = \$dn^2/2$. Increasing the number of oracles is more effective than increasing $\$d$, of course, and can have a dramatic effect: a network with $n = 300$ nodes and deposits $\$d = \$20K$ would be protected against a briber with budget up to $\$900M$.

Note that a staking system can in many cases protect smart contracts representing more value than the offered level of bribery protection. This is because an adversary attacking these contracts cannot extract the full value in many cases. For example, a Chainlink-powered contract securing $\$1B$ in value may only require security against a briber with $\$100M$ in resource because such an adversary can feasibly extract a profit of only 10% of the value of the contract.

Note: The idea that the value of a network can grow quadratically is expressed in the well known Metcalfe's Law [167, 235], which states that the value of a network grows quadratically in the number of connected entities. Metcalfe's Law, however, arises from growth in the number of potential pairwise network connections, a different phenomenon than that underlying quadratic staking impact in our incentive mechanism.

9.4.3 Realization of Second Tier

Two operational features facilitate realization of a high-reliability second tier: (1) Second-tier adjudication should be a rare event in oracle networks and therefore can be significantly more costly than normal operation of the first tier and (2) Assuming

optimistically accepted reports—or contracts whose execution can await arbitration—the second tier need not execute in real time. These features result in a range of configuration options for the second tier to meet the requirements of particular DONs.

As an example approach, a second tier committee can consist of nodes selected by a DON (i.e., first tier) from the longest-serving and most reliable nodes in the Chainlink network. In addition to considerable relevant operational experience, the operators of such nodes have a considerable implicit incentive in FFO that motivates a desire to ensure that the Chainlink network remains highly reliable. They also have publicly available performance histories that provide transparency into their reliability. Second-tier nodes, it is worth noting, need not be participants in the first-tier network, and may adjudicate faults across multiple first-tier networks.

Nodes in a given DON can pre-designate and publicly commit to a set of n' such nodes as constituting the second-tier committee for that DON. Additionally, DON nodes publish a parameter $k' \leq n'$ that determines the number of second-tier votes required to penalize a first-tier node. When an alert is generated for a given report, the members of the second tier vote on the correctness of the values provided by each of the first-tier nodes. Any first-tier node that receives k' negative votes forfeits its deposits to the watchdog node.

Because of the rareness of adjudication and opportunity for extended-time execution noted above, in contrast to the first tier, nodes in the second tier can:

1. Be highly compensated for conducting adjudication.

2. Draw on additional data sources, beyond even the diverse set used by the first-tier.

3. Rely on manual and/or expert inspection and intervention, e.g., to identify and reconcile errors in source data and distinguish between an honest node relaying faulty data and a misbehaving node.

We emphasize that the approach we have just described for selection of second-tier nodes and policy governing adjudication represents just a point within a large design space of possible realizations of the second tier. Our incentive mechanism offers *complete flexibility as to how the second tier is realized*. Individual DONs can thus constitute and set rules for their second tiers that meet the particular requirements and expectations of participating nodes and users.

DECO and Town Crier as adjudication tools: It is essential for the second tier in our mechanism to be able to distinguish between adversarial first-tier nodes that intentionally produce incorrect reports and honest first-tier nodes that unintentionally relay data that is incorrect at the source. Only then can the second tier implement slashing to disincentivize cheating, the goal of our mechanism. DECO and Town Crier are powerful tools that can enable second-tier nodes to make this critical distinction reliably.

Second-tier nodes may in some cases be able to directly query the data source used by a first-tier node or use ADO Section 7.1 in order to check whether an incorrect report resulted from a faulty data source. In other cases, however, second tier nodes may lack direct access to a first-tier node's data source. In such cases, correct adjudication would appear to be infeasible or require a reliance on subjective judgment. Previous oracle dispute systems have relied upon inefficient, escalating rounds of voting to address such challenges.

Using DECO or Town Crier, however, a first tier node can prove correct behavior to second-tier nodes. (See Section 3.6.2 for details on the two systems.) Specifically, if the second tier node identifies a first-tier node as having output a faulty report value $\tilde{r}$, the first-tier node can use DECO or Town Crier to generate tamperproof evidence for second-tier nodes that it is correctly relaying $\tilde{r}$ correctly from a (TLS-enabled) source recognized as authoritative by the DON. Critically, the first-tier node can do this *without second-tier nodes requiring direct access to the data source.*[17] Consequently, correct adjudication is feasible in Chainlink for any desired data source.

9.4.4 Misreporting Insurance

The strong bribery resistance achieved by our staking mechanism relies fundamentally on slashed funds *being awarded to alerters*. Without a monetary reward, alerters would have no direct incentive to reject bribes. As a result, however, slashed funds are not available to compensate users harmed by incorrect reports, e.g., users that lose money when incorrect price data is relayed to a smart contract.

By assumption, incorrect reports don't pose a problem if reports are accepted by a contract only after potential adjudication, i.e., action by the second tier. As explained above, though, to achieve the best possible performance, contracts may instead rely *optimistically* on the mechanism to enforce correct reporting, meaning that they accept reports *before* potential second-tier adjudication. Indeed, such optimistic behavior is safe in our model assuming rational adversaries whose budgets do not exceed the staking impact of the mechanism.

Users concerned about the improbable event of a mechanism failure resulting from, e.g., adversaries with overwhelming financial resources, may wish to employ an additional layer of economic security in the form of *misreporting insurance*. We know of multiple insurers already intending to offer smart-contract-backed policies of this kind for Chainlink-secured protocols in the near future, including through innovative mechanisms such as DAOs, e.g., [7]. The existence of performance history for Chainlink nodes and other data about nodes such as their stake amounts provides an exceptionally strong basis for actuarial assessments of risk, making it possible to price policies in ways that are inexpensive for policyholders yet sustainable for insurers.

[17] With Town Crier, it is additionally possible for first-tier nodes to locally generate attestations of correctness for the reports they output and provide these attestations to second-tier nodes on an as-needed basis.

Basic forms of misreporting insurance can be implemented in a trustworthy and efficient manner using smart contracts. As a simple example, a parametric insurance contract $\mathsf{SC}_{\mathsf{ins}}$ can compensate policyholders automatically if our incentive mechanism's second tier identifies an error in a report generated in the first tier.

A user $\mathcal{U}$ that wishes to purchase an insurance policy, e.g., the creator of a target contract SC, can submit a request to a decentralized insurer for an policy amount $\$M$ on the contract. On approving $\mathcal{U}$, the insurer can set an ongoing (e.g., monthly) premium of $\$P$ in $\mathsf{SC}_{\mathsf{ins}}$. While $\mathcal{U}$ pays the premium, her policy remains active.

If a reporting failure occurs in SC, the result will be the emission of a pair (r_1, r_2) of conflicting reports for SC, where r_1 is signed by the first tier in our mechanism and r_2, the corresponding corrected report, is signed by the second tier. If the $\mathcal{U}$ furnishes such a valid pair (r_1, r_2) to $\mathsf{SC}_{\mathsf{ins}}$, the contract automatically pays her $\$M$, provided her premium payments are up-to-date.

9.5 Single-Round Variant

The protocol described in the previous subsection requires that the second-tier committee wait n rounds to determine whether a watchdog has raised an alert. This requirement holds even in the optimistic case, i.e., when the first tier is functioning correctly. For users unwilling to accept reports optimistically, i.e., prior to potential adjudication, the delay associated with that approach would be unworkable.

For this reason, we are also exploring alternative protocols that require just one round. In this approach, all oracle nodes submit secret bits indicating whether or not they wish to raise an alert. The second-tier committee then checks these values in priority order. To provide a rough sketch, such a scheme might involve the following steps:

1. *Watchdog bit submission:* Each node $\mathcal{O}_i$ *secret-shares* a one-bit watchdog value $w_i \in \{\texttt{no\_alert}, \texttt{alert}\}$ among nodes in the second tier for every report it generates.

2. *Anonymous tips:* Any oracle node can submit an *anonymous tip* α to the second-tier committee in the same round that watchdog bits are submitted. This tip α is a message indicating that an alert has been raised for the current report.

3. *Watchdog bit checking:* The second-tier committee reveals oracle nodes' watchdog bits in priority order.

Note that nodes must send `no_alert` watchdog bits when they don't alert: otherwise, traffic analysis reveals all nodes' bits. The protocol *does* reveal the `no_alert` watchdog bits of nodes with higher priority than the highest-priority alerting watchdog.

Observe that what is revealed is identical with that of our n-round protocol. Rewards are also distributed identically with that scheme, i.e., the first identified watchdog receives the slashed deposits of nodes that have submitted incorrect reports.

The use of anonymous tips enables the second-tier committee to remain *non-interactive* in cases where no alert has been raised, reducing communication complexity in the common case. Note that any watchdog that raises an alert has an economic incentive to submit an anonymous tip: If no tip is submitted, no reward is paid to any node.

To ensure that the sender $\mathcal{O}_i$ of an anonymous tip α cannot be identified by the adversary based on network data, the anonymous tip can be sent over an *anonymous channel*, e.g, via Tor, or, more practically, proxied via a cloud service provider. To authenticate the tip as originating with $\mathcal{O}$, $\mathcal{O}_i$ can sign α using a *ring signature* [39, 192]. Alternatively, to prevent unattributable denial-of-service attacks against the second-tier committee by a malicious oracle node, α can be an anonymous credential with revocable anonymity [73].

This protocol, while practically achievable, has somewhat heavyweight engineering requirements (which we are exploring ways to reduce). First-tier nodes, for instance, must communicate directly with second-tier nodes, requiring maintenance of a directory. The need for anonymous channels and ring signatures adds to the engineering complexity of the scheme. Finally, there is a special trust requirement briefly discussed in the note below. We are therefore also exploring simpler schemes that still achieve super-linear staking impact, but perhaps less than quadratic, in which a briber asymptotically needs resources of at least $\$n \log n$, for example. Some of the schemes under consideration involve random selection of a strict subset of nodes to act as watchdogs, in which case prospective bribery becomes an especially powerful attack.

Remark: The security of this single-round staking mechanism requires *untappable channels* between oracle and second-tier nodes—a standard requirement in coercion-resistant systems, e.g., voting [82, 138], and a reasonable one in practice.

Additionally, however, a node $\mathcal{O}_i$ that seeks to cooperate with a briber can construct its secret shares in such a way as to show the briber that it has encoded a particular value. For example, if $\mathcal{O}_i$ does not know which nodes the briber controls, then $\mathcal{O}_i$ can submit 0-valued shares to all committee members. The briber can then verify $\mathcal{O}_i$'s compliance probabilistically. To avoid this problem in any single-round protocol, we require that $\mathcal{O}_i$ know the identity of at least one honest second-tier node.

With an interactive protocol in which each second-tier node adds a randomization factor to shares, the best the briber can do is enforce selection by $\mathcal{O}_i$ of a random watchdog bit.

9.6 Implicit-Incentive Framework (IIF)

FFO is a form of implicit incentive for correct behavior in the Chainlink network. It functions like explicit stake, i.e., deposits, in that it helps enforce economic security for the network. In other words, FFO should be included as part of the (effective) deposit $\$d$ of a node in the network.

The question is: How do we measure FFO and other forms of implicit incentive within the Chainlink network? The Implicit-Incentive Framework (IIF) is a set of principles and techniques that we plan to develop for this purpose. Blockchain systems provide many forms of unprecedented transparency, and the high-trust records of node performance they create are a springboard for our vision of how the IIF will work. Here we very briefly sketch ideas on key elements of the IIF.

The IIF itself will consist of a set of factors we identify as important in evaluating implicit incentives, along with mechanisms for publishing relevant data in a high-assurance form for consumption by analytics algorithms. Different Chainlink users may wish to use the IIF in different ways, e.g., giving different weighting to different factors. We expect analytics services to arise in the community that help users apply the IIF according to their individual risk-evaluation preferences, and our goal is to facilitate such services by ensuring their access to high-assurance and timely supporting data, as we discuss below (Section 9.6.4).

9.6.1 Future Fee Opportunity

Nodes participate in the Chainlink ecosystem to earn a share of the fees that the networks pay out for any of the various services we have described in this paper, from ordinary data feeds to advanced services such as decentralized identity, fair sequencing, and confidentiality-preserving DeFi. Fees in the Chainlink network support node operators' costs for, e.g., running servers, acquiring necessary data licenses, and maintaining a global staff to ensure high uptime. FFO denotes the service fees, net of expenses, that a node stands to gain in the future—or lose should it demonstrate faulty behavior. FFO is a form of stake that helps secure the network.

A helpful feature of FFO is the fact that on-chain data (supplemented by off-chain data) establish a high-trust record of a node's history, enabling computation of FFO in a transparent, *empirically driven* manner.

A simple, first-order measure of FFO can derive from the average net revenue of a node over a period of time (i.e., gross revenue minus operating expenses). FFO may then be calculated as, e.g., the *net present value* [114] of cumulative future net revenue, in other words, the time-discounted value of all future earnings.

Node revenue can be volatile, however, as shown for example in Fig. 17.

More importantly, node revenue may not follow a distribution that is stationary over time. Consequently, other factors we plan to explore in estimating FFO include:

- *Performance history:* An operator's performance history—including the correctness and timeliness of its reports, as well as its up time—provides an objective touchstone for users to evaluate its reliability. Performance history will thus provide a critical factor in users' selection of oracle nodes (or, with the advent of DONs, their selection of DONs). A strong performance history is likely to correlate with high ongoing revenue.[18]

[18] An important research question we intend to address is detection of *falsified* service volumes.

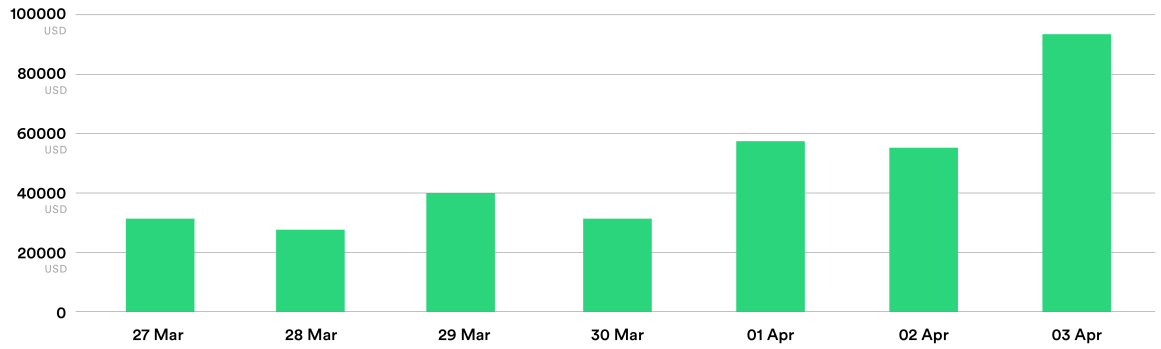

Figure 17: Revenue earned by Chainlink nodes on a single data feed (ETH-USD) during a representative week in March 2021.

- *Data access:* While oracles may obtain many forms of data from open APIs, certain forms of data or certain high-quality sources may be available only on a subscription basis or through contractual agreements. Privileged access to certain data sources can play a role in creating a stable revenue stream.
- *DON participation:* With the advent of DONs, communities of nodes will come together to provide particular services. We expect that many DONs will include operators on a selective basis, establishing participation in reputable DONs as a privileged market position that helps ensure a consistent source of revenue.
- *Cross-platform activity:* Some node operators may have well-established presences and performance track records in other contexts, e.g., as PoS validators or data providers in non-blockchain contexts. Their performance in these other systems (when data on it is available in a trustworthy form) can inform evaluation of their performance history. Similarly, faulty behavior in the Chainlink network can jeopardize revenue in these other systems by driving away users, i.e., FFO can extend across platforms.

9.6.2 Speculative FFO

Node operators participate in the Chainlink network not just to generate revenue from operations, but to create and position themselves to *take advantage of new opportunities to run jobs*. In other words, expenditure by oracle nodes in the network is also a positive statement about the future of DeFi and other smart-contract application domains as well as emerging non-blockchain applications of oracle networks. Node operators today earn the fees available on existing Chainlink networks and simultaneously

These are loosely analogous to fake reviews on internet sites, except that the problem is easier in the oracle setting because we have a definitive record of whether the goods, i.e., reports, were ordered and delivered—as opposed to, e.g., physical goods ordered in online shops. Put another way, in the oracle setting, performance can be validated, even if customer veracity can't.

build a reputation, performance history, and operational expertise that will position them advantageously to earn fees available in future networks (contingent, of course, on honest behavior). The nodes operating in the Chainlink ecosystem today will in this sense have an advantage over newcomers in earning the fees as additional Chainlink services become available. This advantage applies to new operators, as well as technology companies with established reputations; for example, T-Systems, a traditional technology provider (subsidiary of Deutsche Telekom), and Kraken, a large centralized exchange, have established early presences in the Chainlink ecosystem [28, 143].

Such participation by oracle nodes in future opportunities may be regarded itself as a kind of *speculative* FFO, and thus constitutes a form of stake in the Chainlink network.

9.6.3 External Reputation

The IIF as we have described it can operate in a network with *strictly pseudonymous operators*, i.e., without disclosure of the people or real-world entities involved.

One potentially important factor for user selection of providers, however, is *external reputation*. By external reputation, we mean the perception of trustworthiness attaching to *real-world identities*, rather than pseudonyms. Reputational risk attaching to real-world identities can be viewed as a form of implicit incentive. We view reputation through the lens of the IIF, i.e., in a cryptoeconomic sense, as a means of establishing cross-platform activity that may be incorporated into FFO estimates.

The benefit of using external reputation as a factor in estimates of FFO, as opposed to pseudonymous linkage, is that external reputation links performance not just to an operator's existing activities, but also to future ones. If, for instance, a bad reputation attaches to an individual person, it can taint that person's future enterprises. Put another way, external reputation can capture a broader swath of FFO than pseudonymous performance records, as the impact of malfeasance attaching to a person or established company is harder to escape than that associated with a pseudonymous operation.

Chainlink is compatible with decentralized identity technologies (Section 4.3) that can provide support for the use of external reputation in the IIF. Such technologies can validate and thereby help ensure the veracity of operators' asserted real-world identities.[19]

9.6.4 Open IIF Analytics

The IIF, as we have noted, aims to provide reliable open-source data and tools for implicit-incentive analytics. The goal is to enable providers within the community to develop analytics tailored to the risk-assessment needs of different parts of the Chainlink user base.

[19] Decentralized identity credentials can also, where desired, embellish pseudonyms with validated supplementary information. For example, a node operator could in principle use such credentials to prove that it is a Fortune 500 company, without revealing which one.

A considerable amount of historical data regarding nodes' revenue and performance resides on chain in a high-trust, immutable form. Our goal, however, is to provide the most comprehensive possible data, including data on behaviors that are visible only off chain, such as Off-Chain Reporting (OCR) or DON activity. Such data can potentially be voluminous. The best way to store it and ensure its integrity, i.e., protect it from tampering, we believe, will be with the help of DONs, using techniques discussed in Section 3.3.

Some incentives lend themselves to direct forms of measurement, such as staking deposits and basic FFO. Others, such as speculative FFO and reputation, are harder to measure in an objective manner, but we believe that supporting forms of data, including historical growth of the Chainlink ecosystem, social-media metrics of reputation, etc., can support IIF analytics models even for these harder-to-quantify elements.

We can imagine that dedicated DONs arise specifically to monitor, validate, and record data relating to off-chain performance records of nodes, as well as other data used in the IIF, such as validated identity information. These DONs can provide uniform, high-trust IIF data for any analytics providers serving the Chainlink community. They will also provide a golden record that makes the claims of analytics providers independently verifiable by the community.

9.7 Putting It All Together: Node Operator Incentives

Synthesizing our discussions above on explicit and implicit incentives for node operators provides a holistic view of the ways that node operators participate in and benefit from the Chainlink network.

As a conceptual guide, we can express the total assets at stake by a given Chainlink node operator $\$S$ in a rough, stylized form as:

$$\$S \approx \$D + \$F + \$FS + \$R,$$

where:

- $\$D$ is the aggregate of all explicitly deposited stake across all networks in which the operator participates;
- $\$F$ is the net present value of the aggregate of all FFO across all networks in which the operator participates;
- $\$FS$ is the net present value of the speculative FFO of the operator; and
- $\$R$ is the reputational equity of the operator outside the Chainlink ecosystem that might be jeopardized by identified misbehavior in its oracle nodes.

While largely conceptual, this rough equality helpfully shows that there is a multiplicity of economic factors favoring high-reliability performance by Chainlink nodes. All of these factors other than $\$D$ are present in today's Chainlink networks.

9.8 The Virtuous Cycle of Economic Security

The combination of super-linear staking impact with representation of fee payments as future fee opportunity (FFO) in the IIF can lead to what we call the *virtuous cycle of economic security* in an oracle network. This can be seen as a kind of *economy of scale*. As the total amount secured by a particular network rises, the amount of additional stake it takes to add a fixed amount of economic security decreases as does the average per-user cost. It's therefore cheaper, in terms of fees, for a user to join an already-existing network than to achieve the same increase in network economic security by creating a new network. Importantly, the addition of each new user lowers the cost of the service for all previous users of that network.

Given a particular fee structure (e.g. a particular yield rate on the amount staked), if the total fees earned by a network increases, this incentivizes the flow of additional stake into the network to secure it at a higher rate. Specifically, if the total stake an individual node may hold in the system is capped, then when new fee payments enter the system, raising its FFO, the number of nodes n will increase. Thanks to the super-linear staking impact of our incentive system design, the economic security of the system will rise faster than n, e.g., as n^2 in the mechanism we sketch in Section 9.4. As a result, the average cost for economic security—i.e., amount of stake contributing a dollar of economic security—will drop. The network can therefore charge its users lower fees. Assuming that demand for oracle services is *elastic* (see, e.g., [31] for a brief explanation), demand will rise, generating additional fees and FFO.

We illustrate this point with the following example.

Example 5. *Since the economic security of an oracle network with our incentive scheme is $\$dn^2$ for stake $\$dn$, the economic security contributed by a dollar of stake is n and thus the average cost per dollar of economic security—i.e, amount of stake contributing to a dollar of economic security—is $1/n$.*

Consider a network in which the economic incentives consist entirely of FFO, capped at $\$d \leq \$10K$ per node. Suppose the network has $n = 3$ nodes. Then the average cost per dollar of economic security is about $\$0.33$.

Suppose that the total FFO of the network rises above $\$30K$ (e.g., to $\$31K$). Given the cap on per-node FFO, the network grows to (at least) $n = 4$. Now the average cost per dollar of economic security drops to about $\$0.25$.

We illustrate the full virtuous cycle of economic security in oracle networks schematically in Fig. 18.

We emphasize that the virtuous cycle of economic security derives from the effect of users *pooling their fees*. It is their collective FFO that works in favor of larger network sizes and thus greater collective security. We also note that the virtuous cycle of economic security works in favor of DONs achieving *financial sustainability*. Once created, DONs that address user needs should grow to and beyond the point at which revenue from fees exceed operational costs for oracle nodes.

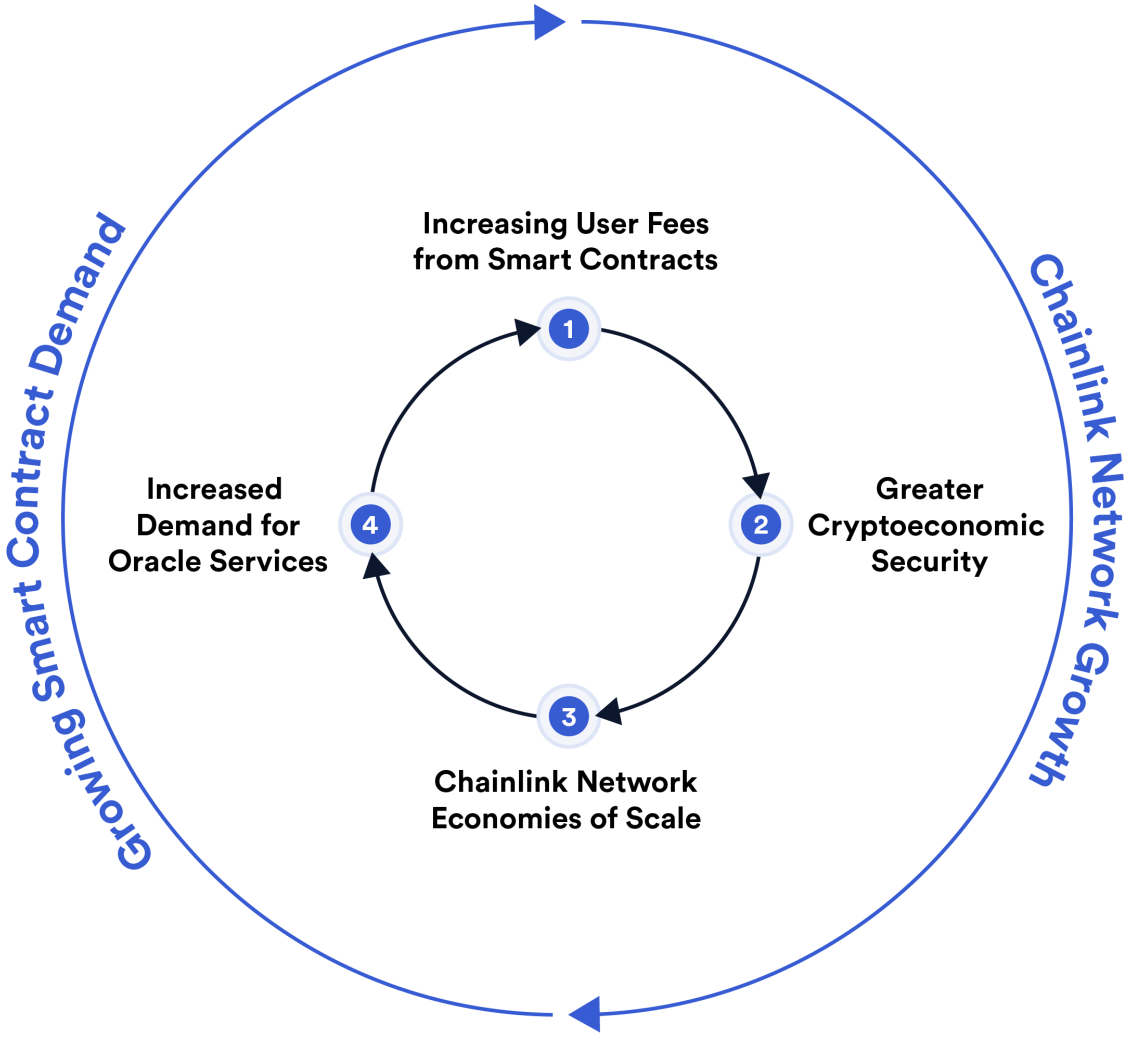

Figure 18: Schematic of the virtuous cycle of Chainlink staking. A rise in user fee payments to an oracle network ① causes it to grow, leading to growth in its economic security ②. This super-linear growth realizes economies of scale in Chainlink networks ③. Specifically, it means a reduction in the average cost of economic security , i.e., the per-dollar economic security arising from fee payments or other sources of stake increases. Lower costs, passed along to users, stimulate increased demand for oracle services ④.

9.9 Additional Factors Driving Network Growth

As the Chainlink ecosystem continues to expand, we believe that its attractiveness to users and importance as infrastructure for the blockchain economy will accelerate. The value provided by oracle networks is super-linear, meaning that it grows faster

than the size of the networks themselves. This growth in value derives from both economies of scale—greater per-user cost efficiency as service volumes increase—and network effects—an increase of network utility as users adopt DONs more widely.

As existing smart contracts continue to see more value secured and entirely new smart contract applications are made possible by more decentralized services, the total use of and aggregate fees paid to DONs should grow. Increasing pools of fees in turn translate into the means and incentive to create even more decentralized services, resulting in a virtuous cycle. This virtuous cycle solves a critical chicken-and-egg problem in the hybrid smart contract ecosystem: Innovative smart contract features often require decentralized services that don't yet exist (e.g., new DeFi markets often require new data feeds) yet need sufficient economic demand to come into existence. The pooling of fees by various smart contracts for existing DONs will signal demand for additional decentralized services from a growing user base, giving rise to their creation by DONs and an ongoing enablement of new and varied hybrid smart contracts.

In summary, we believe that the growth in network security driven by virtuous cycles in the Chainlink staking mechanism exemplifies larger patterns of growth that the Chainlink network can help bring about in an on-chain economy for decentralized services.

10 Conclusion

In this paper, we have set forth a vision for Chainlink's evolution. The main theme in this vision is oracle networks' ability to provide a much broader range of service for smart contracts than mere data delivery. Using DONs as a foundation for the decentralized services of the future, Chainlink will aim to provide performant, confidentiality-enhanced oracle functionality. Its oracle networks will offer strong trust minimization through a combination of principled cryptoeconomic mechanisms such as staking and carefully conceived guard rails and service-level enforcement on relying main chains. DONs will also help layer-2 systems enforce flexible, fair ordering policies on transactions, as well as reduced gas costs for mempool-routed transactions. Taken together, these capabilities all drive in the direction of secure and richly functional hybrid smart contracts.

The flexibility of DONs will enhance existing Chainlink services and give rise to many additional smart contract features and applications. Among these are seamless connection to a wide variety of off-chain systems, decentralized identity creation from existing data, priority channels to help ensure timely delivery of infrastructure-critical transactions, and confidentiality-preserving DeFi instruments.

The vision we've set forth here is ambitious. In the short term, we seek to empower hybrid contracts to accomplish goals beyond the reach of smart contracts today, while in the long term we aim to realize a decentralized metalayer. Happily we can draw on new tools and ideas—ranging from consensus algorithms to zero-knowledge proof systems—that the community is developing as the fruit of rapidly evolving research.

Similarly, we expect to prioritize implementation of the ideas in this paper in response to the needs of Chainlink's community of users. We look forward to the next stage in our quest to empower smart contracts through universal connectivity and establish decentralized technologies as the backbone of the world's next generation of financial and legal systems.

Acknowledgements

Thanks to Julian Alterini and Shawn Lee for rendering the figures in this paper.

References

[1] DeFi pulse. https://defipulse.com. [Online; accessed 30 Mar. 2021].

[2] dYdX. dydx.exchange. [Online; accessed 30 Mar. 2021].

[3] Ethereum WebAssembly: Metering. https://ewasm.readthedocs.io/en/mkdocs/metering. [Online; accessed 30 Mar. 2021].

[4] Keybase: End-to-end encryption for things that matter. keybase.io. [Online; accessed 30 Mar. 2021].

[5] Loopring: zkRollup exchange and payment protocol. https://loopring.org. [Online; accessed 30 Mar. 2021].

[6] MetaMask: A crypto wallet and gateway to blockchain apps. https://metamask.io. [Online; accessed 30 Mar. 2021].

[7] Nexus Mutual: a people-powered alternative to insurance. nexus.io. [Online; accessed 30 Mar. 2021].

[8] Oasis labs. https://oasislabs.com. [Online; accessed 30 Mar. 2021].

[9] OpenID connect authentication. https://openid.net/connect. [Online; accessed 30 Mar. 2021].

[10] Optimism. https://optimism.io. [Online; accessed 30 Mar. 2021].

[11] Optimistic Rollups - EthHub. https://docs.ethhub.io/ethereum-roadmap/layer-2-scaling/optimistic_rollups. [Online; accessed 30 Mar. 2021].

[12] Starkware: bringing scalability and privacy to a blockchain near you. starkware.co. [Online; accessed 30 Mar. 2021].

[13] Tornado Cash. https://tornado.cash. [Online; accessed 30 Mar. 2021].

[14] uPort: Open identity system for the decentralized web. https://www.uport.me. [Online; accessed 30 Mar. 2021].

[15] Wrapped Bitcoin: WBTC. wbtc.network. [Online; accessed 30 Mar. 2021].

[16] zkSync: Secure, scalable crypto payments. https://zksync.io. [Online; accessed 30 Mar. 2021].

[17] Introduction to emergency shutdown in multi-collateral Dai. *MakerDAO blog*. https://blog.makerdao.com/introduction-to-emergency-shutdown-in-multi-collateral-dai, 12 August 2019.

[18] Chainlink announcement: Introducing the Chainlink on-chain data directory: Data.eth. *Chainlink Blog.* https://blog.chain.link/introducing-the-chainlink-on-chain-data-directory, 28 Dec. 2020.

[19] Speed bump on former AMEX exchange made spreads worse, NYSE say. S&P Global Market Intelligence, https://www.spglobal.com/marketintelligence/en/news-insights/trending/O1_i8QswCPPwWV9HbS-5dw2, 4 Nov. 2019.

[20] Chainlink developers documentation: Adapters. https://docs.chain.link/docs/adapters, [Online; accessed 30 Mar. 2021].

[21] Chainlink developers documentation: Initiators. https://docs.chain.link/docs/initiators, [Online; accessed 30 Mar. 2021].

[22] Ethereum name service: Decentralized naming for wallets, websites, & more. https://ens.domains, [Online; accessed 30 Mar. 2021].

[23] Aavegotchi Wiki contributors. Aavegotchi: Introduction — Aavegotchi Wiki, 2021. [Online; accessed 30 Mar. 2021].

[24] Abe, M., and Fehr, S. Adaptively secure Feldman VSS and applications to universally-composable threshold cryptography. In *Advances in Cryptology (CRYPTO)* (2004), pp. 317–334.

[25] Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., and Kastania, A. Astraea: A decentralized blockchain oracle. In *IEEE iThings / GreenCom / CPSCom / SmartData* (2018), pp. 1145–1152.

[26] Adler, J., Berryhill, R., Veneris, A. G., Poulos, Z., Veira, N., and Kastania, A. Astraea: A decentralized blockchain oracle. In *2018 IEEE International Conference on Internet of Things (iThings)* (2018).

[27] Ahmad, A., Joe, B., Xiao, Y., Zhang, Y., Shin, I., and Lee, B. Obfuscuro: A commodity obfuscation engine on Intel SGX. In *Networks and Distributed Security Systems (NDSS)* (2019).

[28] Akhtar, T. Kraken Exchange brings its spot price data to DeFi via new Chainlink node. *Coindesk* (1 Feb. 2021). https://www.coindesk.com/kraken-exchange-brings-its-spot-price-data-to-defi-via-new-chainlink-node.

[29] Ali, M., Nelson, J., Shea, R., and Freedman, M. J. Blockstack: A global naming and storage system secured by blockchains. In *USENIX Annual Technical Conference (ATC)* (2016).

[30] Allen, S., Čapkun, S., Eyal, I., Fanti, G., Ford, B., Grimmelmann, J., Juels, A., Kostiainen, K., Meiklejohn, S., Miller, A., et al. Design choices for central bank digital currency: Policy and technical considerations. *NBER Working Paper Series*, Working paper 27634 (Aug. 2020).

[31] Anderson, P. L., McLellan, R. D., Overton, J. P., and Wolfram, G. L. Price elasticity of demand. *McKinac Center for Public Policy 13* (1997), 2010.

[32] Arnosti, N., and Weinberg, S. M. Bitcoin: A natural oligopoly. *arXiv preprint arXiv:1811.08572* (2018).

[33] Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., and Yakira, D. Helix: a scalable and fair consensus algorithm. Tech. rep., Technical report, Orbs Research, 2018.

[34] Attah, E. Five most prolific 51% attacks in crypto: Verge, Ethereum Classic, Bitcoin Gold, Feathercoin, Vertcoin. *CryptoSlate* (24 April 2019).

[35] Axie Infinity. Axie Infinity integrates Chainlink oracles! *Axie Infinity blog.* https://axieinfinity.medium.com/axie-infinity-integrates-chainlink-oracles-aa93d3d0983e, 16 Nov. 2020.

[36] Baird, L., Luykx, A., and Madsen, P. Hedera technical insights: Fair timestamping and fair ordering of transactions. *Hedera Blog.* https://hedera.com/blog/fair-timestamping-and-fair-ordering-of-transactions, 12 Apr. 2020.

[37] Baum, C., Orsini, E., Scholl, P., and Soria-Vazquez, E. Efficient constant-round MPC with identifiable abort and public verifiability. In *Advances in Cryptology (CRYPTO)* (2020), pp. 562–592.

[38] Ben-Sasson, E., Bentov, I., Horesh, Y., and Riabzev, M. Scalable zero knowledge with no trusted setup. In *Advances in Cryptology (CRYPTO)* (2019), pp. 701–732.

[39] Bender, A., Katz, J., and Morselli, R. Ring signatures: Stronger definitions, and constructions without random oracles. *Journal of Cryptology 22*, 1 (2009), 114–138.

[40] Benet, J. IPFS-content addressed, versioned, P2P file system. *arXiv preprint arXiv:1407.3561* (2014).

[41] Benhamouda, F., Gentry, C., Gorbunov, S., Halevi, S., Krawczyk, H., Lin, C., Rabin, T., and Reyzin, L. Can a blockchain keep a secret? *IACR Cryptol. ePrint Arch. 2020* (2020), 464.

[42] Benhamouda, F., Halevi, S., and Halevi, T. Supporting private data on Hyperledger Fabric with secure multiparty computation. *IBM Journal of Research and Development 63*, 2/3 (2019), 3–1.

[43] Benson, J. ConsenSys wades into compliance for Ethereum tokens. https://decrypt.co/31641/consensys-wades-into-compliance-for-ethereum-tokens, 8 June 2020. [Online; accessed 30 Mar. 2021].

[44] Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P., and Juels, A. Tesseract: Real-time cryptocurrency exchange using trusted hardware. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2019), pp. 1521–1538.

[45] Berberich, M., and Steiner, M. Blockchain technology and the GDPR-how to reconcile privacy and distributed ledgers. *Eur. Data Prot. L. Rev. 2* (2016), 422.

[46] Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., et al. Formal verification of smart contracts: Short paper. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security* (2016), pp. 91–96.

[47] Birman, K. P., and Schneider, F. B. The monoculture risk put into context. *IEEE Security & Privacy 7*, 1 (2009), 14–17.

[48] Blocknative. Evidence of mempool manipulation on Black Thursday: Hammerbots, mempool compression, and spontaneous stuck transactions. *Blocknative Blog.* https://blog.blocknative.com/blog/mempool-forensics, 22 July 2020.

[49] Bloemenm, R., Logvinov, L., and Evans, J. Ethereum Improvement Proposal (EIP) 712: Ethereum typed structured data hashing and signing. https://eips.ethereum.org/EIPS/eip-712, 12 Sept. 2017.

[50] Bogatyy, I. Implementing Ethereum trading front-runs on the Bancor exchange in Python. *Hackernoon.* https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798, 17 Aug. 2017.

[51] Böhme, R., Christin, N., Edelman, B., and Moore, T. Bitcoin: Economics, technology, and governance. *Journal of Economic Perspectives 29*, 2 (2015), 213–38.

[52] Bojja Venkatakrishnan, S., Fanti, G., and Viswanath, P. Dandelion: Redesigning the Bitcoin network for anonymity. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 1*, 1 (2017), 1–34.

[53] Boneh, D., Bonneau, J., Bünz, B., and Fisch, B. Verifiable delay functions. In *Advances in Cryptology (CRYPTO)* (2018), pp. 757–788.

[54] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the Weil pairing. In *Advances in Cryptology (ASIACRYPT)* (2001), pp. 514–532.

[55] Boneh, D., Lynn, B., and Shacham, H. Short signatures from the weil pairing. In *ASIACRYPT* (2001), vol. 2248 of *Lecture Notes in Computer Science*, Springer, pp. 514–532.

[56] Bootle, J., Cerulli, A., Chaidos, P., Groth, J., and Petit, C. Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting. In *Advances in Cryptology (EUROCRYPT)* (2016), pp. 327–357.

[57] Bowers, K. D., Juels, A., and Oprea, A. HAIL: A high-availability and integrity layer for cloud storage. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2009), pp. 187–198.

[58] Bowman, M., Miele, A., Steiner, M., and Vavala, B. Private data objects: an overview. *arXiv preprint arXiv:1807.05686* (2018).

[59] Boyle, E., Goldwasser, S., and Ivan, I. Functional signatures and pseudorandom functions. In *Public Key Cryptography (PKC)* (2014), pp. 501–519.

[60] Breidenbach, L., Cachin, C., Coventry, A., Juels, A., and Miller, A. Chainlink off-chain reporting protocol. https://chain.link/ocrpaper, 2021. [Online; accessed 30 Mar. 2021].

[61] Breidenbach, L., Daian, P., and Tramèr, F. GasToken. gastoken.io. [Online; accessed 30 Mar. 2021].

[62] Breidenbach, L., Daian, P., Tramèr, F., and Juels, A. Enter the Hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *USENIX Security Symposium (USENIX Security)* (2018), pp. 1335–1352.

[63] Brown, M., and Housley, R. Transport layer security (TLS) evidence extensions, Nov. 2006. Working Draft, IETF Secretariat, Internet-Draft drafthousley-evidence-extns-01.

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

[65] Bünz, B., Agrawal, S., Zamani, M., and Boneh, D. Zether: Towards privacy in a smart contract world. In *Financial Cryptography and Data Security (FC)* (2020), pp. 423–443.

[66] Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., and Maxwell, G. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy (SP)* (2018), IEEE, pp. 315–334.

[67] Buterin, V. The p + epsilon attack. *Ethereum Blog*. https://blog.ethereum.org/2015/01/28/p-epsilon-attack, 28 Jan. 2015.

[68] Buterin, V. SchellingCoin: A Minimal-Trust Universal Data Feed. *Ethereum Blog*. https://blog.ethereum.org/2014/03/28/schellingcoin-a-minimal-trust-universal-data-feed, 28 Mar. 2014.

[69] Buterin, V. On-chain scaling to potentially ~500 tx/sec through mass tx validation. *Ethereum Blog*. https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477, 3 Sept. 2018.

[70] Buterin, V., Dietrichs, A., Garnett, M., Villanueva, W., and Wilson, S. Ethereum Improvement Proposal (EIP) 2938: Account abstraction. https://eips.ethereum.org/EIPS/eip-2938, 4 Sept. 2020.

[71] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. Secure and efficient asynchronous broadcast protocols (extended abstract). In *Advances in Cryptology: CRYPTO 2001* (2001), J. Kilian, Ed., vol. 2139, Springer, pp. 524–541.

[72] Cachin, C., and Vukolić, M. Blockchain consensus protocols in the wild. *arXiv preprint arXiv:1707.01873* (2017).

[73] Camenisch, J., and Lysyanskaya, A. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *International conference on the theory and applications of cryptographic techniques* (2001), pp. 93–118.

[74] Cecchetti, E., Fisch, B., Miers, I., and Juels, A. PIEs: Public incompressible encodings for decentralized storage. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2019), pp. 1351–1367.

[75] Chainlink. How to build a parametric insurance smart contract. *Chainlink Blog*. https://blog.chain.link/parametric-insurance-smart-contract, 15 Dec. 2020.

[76] Chainlink. Chainlink Proof of Reserve: Bringing transparency to DeFi collateral. *Chainlink Blog*. https://blog.chain.link/chainlink-proof-of-reserve-bringing-transparency-to-defi-collateral, 30 Nov. 2020.

[77] Chainlink. How Chainlink supports any off-chain data resource and computation. *Chainlink Blog*. https://blog.chain.link/how-chainlink-supports-any-off-chain-data-resource-and-computation, 8 Mar. 2021.

[78] Chainlink. Introduction to Chainlink VRF. *Chainlink Developers Documentation*. https://docs.chain.link/docs/chainlink-vrf, [Online; accessed 30 Mar. 2021].

[79] Chan, J., Warwick, K., and Ennis, C. Synthetix improvement proposal (SIP) 6: Frontrunning protection. https://sips.synthetix.io/sips/sip-6, 27 June 2019.

[80] Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., and Song, D. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In *IEEE European Symposium on Security and Privacy (EuroS&P)* (2019), IEEE, pp. 185–200.

[81] Chjango Unchained. Tendermint explained — bringing BFT-based PoS to the public blockchain domain. *Cosmos Blog*. https://blog.cosmos.network/tendermint-explained-bringing-bft-based-pos-to-the-public-blockchain-domain-f22e274a0fdb, 10 May 2018.

[82] Clarkson, M. R., Chong, S., and Myers, A. C. Civitas: Toward a secure voting system. In *IEEE Symposium on Security and Privacy (SP)* (2008), IEEE, pp. 354–368.

[83] Cloudflare. Understanding AMP real URL. https://support.cloudflare.com/hc/en-us/articles/360029367652-Understanding-Amp-Real-URL, [Online; accessed 30 Mar. 2021].

[84] Costan, V., and Devadas, S. Intel SGX explained. *IACR Cryptol. ePrint Arch. 2016*, 86 (2016), 1–118.

[85] Crawley, J. Chainlink integration connects Filecoin to smart contract-enabled blockchains. *Coindesk* (24 Mar. 2021). https://www.coindesk.com/filecoin-chainlink-integration-smart-contract-enabled-blockchains.

[86] Croman, K., Decker, C., Eyal, I., Gencer, A. E., Juels, A., Kosba, A., Miller, A., Saxena, P., Shi, E., Sirer, E. G., et al. On scaling decentralized blockchains. In *Financial Cryptography and Data Security (FC)* (2016), pp. 106–125.

[87] Cronje, A. Scaling Keep3r with Chainlink. *Medium Blog Post*. https://andrecronje.medium.com/scaling-keep3r-with-chainlink-2832bbc76506, 2 Dec. 2020.

[88] Crypto51. PoW 51% attack cost. https://www.crypto51.app. [Online; accessed 30 Mar. 2021].

[89] Cusack, L. Improving PoolTogether with Chainlink VRF. *PoolTogether Blog*. https://medium.com/pooltogether/improving-pooltogether-with-chainlink-vrf-dcf1a3d6ea, 11 May 2020.

[90] Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., and Juels, A. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)* (2020), pp. 566–583.

[91] Dale, B. Feature from tech no collateral required: How Aave brought unsecured borrowing to DeFi. *Coindesk* (24 Aug. 2020). https://www.coindesk.com/aave-unsecured-borrowing-defi.

[92] Decentralized Identity Foundation. DIF website. https://identity.foundation. [Online; accessed 30 Mar. 2021].

[93] del Castillo, M. How to track official election results on Ethereum and EOS. *Forbes* (3 Nov. 2020). https://www.forbes.com/sites/michaeldelcastillo/2020/11/03/how-to-track-official-election-results-on-ethereum-and-eos.

[94] Dierks, T., and Rescorla, E. The transport layer security (tls) protocol version 1.2.

[95] Dingledine, R., Mathewson, N., and Syverson, P. Tor: The second-generation onion router. Tech. rep., Naval Research Lab Washington DC, 2004.

[96] Dong, C., Wang, Y., Aldweesh, A., McCorry, P., and van Moorsel, A. Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing. In *ACM Conference on Computer and Communications Security (CCS)* (2017), pp. 211–227.

[97] Duan, S., Reiter, M. K., and Zhang, H. Secure causal atomic broadcast, revisited. In *Proc. 47th International Conference on Dependable Systems and Networks* (2017), pp. 61–72.

[98] Ellis, S., Juels, A., and Nazarov, S. Chainlink: a decentralized oracle network. https://chain.link/whitepaper, 4 Sept. 2017.

[99] Enigma Project. New to enigma? start here. *Enigma Blog*. https://blog.enigma.co/welcome-to-enigma-start-here-e65c8c9125ef, 5 Nov. 2018.

[100] Erik Marks, P. G. Ethereum Improvement Proposal (EIP) 712: Wallet add Ethereum chain RPC method. https://eips.ethereum.org/EIPS/eip-3085, 1 Nov. 2020.

[101] Eskandari, S., Moosavi, S., and Clark, J. SoK: Transparent dishonesty: front-running attacks on blockchain. In *International Conference on Financial Cryptography and Data Security* (2019), pp. 170–189.

[102] EtherCards. The EtherCards platform. https://docs.ether.cards/platform.html. [Online; accessed 30 Mar. 2021].

[103] Ethereum Foundation. Ethereum 2.0 (Eth2). https://ethereum.org/en/eth2, [Online; accessed 30 Mar. 2021].

[104] etherscan. io. Ethereum Average Block Time Chart | Etherscan. https://etherscan.io/chart/blocktime, Sep 2020. [Online; accessed 30 Mar. 2021].

[105] European Data Protection Board. Guidelines 3/2018 on the territorial scope of the GDPR (article 3) [version 2.1]. https://edpb.europa.eu/sites/edpb/files/files/file1/edpb_guidelines_3_2018_territorial_scope_after_public_consultation_en_1.pdf, 12 Nov. 2019.

[106] Eyal, I., and Sirer, E. G. Majority is not enough: Bitcoin mining is vulnerable. In *Financial Cryptography and Data Security (FC)* (2014), pp. 436–454.

[107] Fanti, G., Venkatakrishnan, S. B., Bakshi, S., Denby, B., Bhargava, S., Miller, A., and Viswanath, P. Dandelion++ lightweight cryptocurrency networking with formal anonymity guarantees. *Proceedings of the ACM on Measurement and Analysis of Computing Systems 2*, 2 (2018), 1–35.

[108] Feldman, P. A practical scheme for non-interactive verifiable secret sharing. In *Annual Symposium on Foundations of Computer Science (FOCS)* (1987), IEEE, pp. 427–438.

[109] Felten, E. What's up with Rollup. *Medium*, Offchain Labs. https://medium.com/offchainlabs/whats-up-with-rollup-db8cd93b314e, 18 Nov. 2019.

[110] Felten, E. Front-Running as a Service. *Medium*, Offchain Labs. https://medium.com/offchainlabs/front-running-as-a-service-334c929c945a, 29 June 2020.

[111] Fisch, B. Tight proofs of space and replication. In *Advances in Cryptology (EUROCRYPT)* (2019), pp. 324–348.

[112] Fisch, B., Bonneau, J., Greco, N., and Benet, J. Scaling Proof-of-Replication for Filecoin mining. *Technical report, Stanford University* (2018).

[113] Franz, M. E unibus pluram: massive-scale software diversity as a defense mechanism. In *Proceedings of the 2010 New Security Paradigms Workshop* (2010), pp. 7–16.

[114] Gallo, A. A refresher on net present value. *Harvard Business Review 19* (2014).

[115] Garay, J., Kiayias, A., and Leonardos, N. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques* (2015), pp. 281–310.

[116] Gennaro, R., and Goldfeder, S. Fast multiparty threshold ECDSA with fast trustless setup. In *ACM Conference on Computer and Communications Security* (2018), pp. 1179–1194.

[117] Gennaro, R., and Goldfeder, S. One round threshold ECDSA with identifiable abort. *IACR Cryptol. ePrint Arch. 2020* (2020), 540.

[118] Gennaro, R., Goldfeder, S., and Narayanan, A. Threshold-optimal dsa/ecdsa signatures and an application to bitcoin wallet security. In *International Conference on Applied Cryptography and Network Security* (2016), pp. 156–174.

[119] George, W., and Lesaege, C. An analysis of $p + \epsilon$ attacks on various models of schelling game based systems. In *Cryptoeconomic Systems (CES)* (2020).

[120] Gilad, Y., Hemo, R., Micali, S., Vlachos, G., and Zeldovich, N. Algorand: Scaling byzantine agreements for cryptocurrencies. In *ACM Symposium on Operating Systems Principles (SOSP)* (2017), pp. 51–68.

[121] Gluchowski, A. Evaluating Ethereum L2 Scaling Solutions: A Comparison Framework. *Medium* (Aug 2020).

[122] Goldwasser, S., Ben-Or, M., and Wigderson, A. Completeness theorems for non-cryptographic fault-tolerant distributed computing. In *Symposium on Theory of Computing (STOC)* (1988), pp. 1–10.

[123] Gollmann, D. *Computer security, 3rd edition*. John Wiley & Sons, 2011.

[124] Google. Serve AMP using signed exchanges. https://amp.dev/documentation/guides-and-tutorials/optimize-and-measure/signed-exchange. [Online; accessed 30 Mar. 2021].

[125] Gorokh, A., Banerjee, S., and Iyer, K. When bribes are harmless: The power and limits of collusion-resilient mechanism design. *SSRN*. https://ssrn.com/abstract=3125003, 2019.

[126] Groth, J. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology (CRYPTO)* (2016), pp. 305–326.

[127] Gu, W. C., Raghuvanshi, A., and Boneh, D. Empirical measurements on pricing oracles and decentralized governance for stablecoins. *SSRN 3611231* (2020).

[128] Hu, E. Intentional access delays, market quality, and price discovery: Evidence from IEX becoming an exchange. *SSRN*. https://ssrn.com/abstract=3195001, 15 Mar. 2019.

[129] Hyperledger Project. Hyperledger Indy. https://www.hyperledger.org/use/hyperledger-indy. [Online; accessed 30 Mar. 2021].

[130] Jackson, M. O., and Wilkie, S. Endogenous games and mechanisms: Side payments among players. In *The Review of Economic Studies* (2005), vol. 72, pp. 543–566.

[131] Jansen, M., Hdhili, F., Gouiaa, R., and Qasem, Z. Do smart contract languages need to be Turing complete? In *International Congress on Blockchain and Applications* (2019), pp. 19–26.

[132] Johnson, R., Molnar, D., Song, D., and Wagner, D. Homomorphic signature schemes. In *Cryptographers' Track at the RSA Conference (CT-RSA)* (2002), pp. 244–262.

[133] Johnson, S., Scarlata, V., Rozas, C., Brickell, E., and Mckeen, F. Intel® Software Guard Extensions: EPID provisioning and attestation services, 2016. White Paper.

[134] Joint Task Force. Security and privacy controls for federal information systems and organizations. *NIST Special Publication 800*, 53 Rev. 5 (2020).

[135] Juels, A., Breidenbach, L., Coventry, A., Nazarov, S., Ellis, S., and Magauran, B. Mixicles. https://chain.link/mixicles.pdf, 2019.

[136] Juels, A., Breidenbach, L., Daian, P., Ji, Y., and Tramèr, F. Project Chicago for the study of cryptocurrencies. projectchicago.io. [Online; accessed 30 Mar. 2021].

[137] Juels, A., Breidenbach, L., and Tramèr, F. Fair Sequencing Services: Enabling a provably fair DeFi ecosystem. *Chainlink Blog*. https://blog.chain.link/chainlink-fair-sequencing-services-enabling-a-provably-fair-defi-ecosystem, 11 Sept. 2020.

[138] Juels, A., Catalano, D., and Jakobsson, M. Coercion-resistant electronic elections. In *Towards Trustworthy Elections*. Springer, 2010, pp. 37–63.

[139] Juels, A., Kosba, A., and Shi, E. The ring of Gyges: Investigating the future of criminal smart contracts. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2016), pp. 283–295.

[140] Kaiko. Kaiko partners with Chainlink to bring cryptocurrency market data to smart contracts. *Kaiko Blog.* https://www.kaiko.com/blogs/latest-news/kaiko-partners-with-chainlink-to-bring-cryptocurrency-market-data-to-smart-contracts, 14 Nov. 2018.

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

[142] Kalodner, H. A., Carlsten, M., Ellenbogen, P., Bonneau, J., and Narayanan, A. An empirical study of Namecoin and lessons for decentralized namespace design. In *Workshop on Economics of Information Security (WEIS)* (2015).

[143] Kapilkov, M. Deutsche Telekom's T-Systems is now a Chainlink node operator. *Cointelegraph* (22 July 2020). https://www.coindesk.com/kraken-exchange-brings-its-spot-price-data-to-defi-via-new-chainlink-node.

[144] Kelkar, M., Zhang, F., Goldfeder, S., and Juels, A. Order-fairness for Byzantine consensus. In *Advances in Cryptology (CRYPTO)* (2020), pp. 451–480.

[145] Khalil, R., Gervais, A., and Felley, G. TEX - a securely scalable trustless exchange. *IACR Cryptol. ePrint Arch. 2019* (2019), 265.

[146] Kiayias, A., Russell, A., David, B., and Oliynykov, R. Ouroboros: A provably secure Proof-of-Stake blockchain protocol. In *Advances in Cryptology (CRYPTO)* (2017), pp. 357–388.

[147] Kivlighan, I. The Aave oracle network powered by Chainlink is now live! *Aave Blog.* https://medium.com/aave/the-aave-oracle-network-powered-by-chainlink-is-now-live-45bb8a5a8c4e, 9 Jan. 2020.

[148] Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *IEEE Symposium on Security and Privacy (SP)* (2018), pp. 583–598.

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

[150] Kroll, J. A., Davey, I. C., and Felten, E. W. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Workshop on the Economics of Information Systems (WEIS)* (2013), vol. 2013, p. 11.

[151] Kursawe, K. Wendy, the good little fairness widget. *arXiv preprint arXiv:2007.08303* (2020).

[152] Lee, D., Kohlbrenner, D., Shinde, S., Asanović, K., and Song, D. Keystone: An open framework for architecting trusted execution environments. In *European Conference on Computer Systems (EuroSys)* (2020), pp. 1–16.

[153] Lev-Ari, K., Spiegelman, A., Keidar, I., and Malkhi, D. Fairledger: A fair blockchain protocol for financial institutions. *arXiv preprint arXiv:1906.03819* (2019).

[154] Levi, Y. Bancor's response to today's smart contract vulnerability. *Bancor Blog.* https://blog.bancor.network/bancors-response-to-today-s-smart-contract-vulnerability-dc888c589fe4, 18 June 2020.

[155] Lewis, M. *Flash boys: a Wall Street revolt.* WW Norton & Company, 2014.

[156] Libra Association. Libra whitepaper v2.0. https://libra.org/en-US/white-paper, April 2020.

[157] Lu, D., Yurek, T., Kulshreshtha, S., Govind, R., Kate, A., and Miller, A. HoneyBadgerMPC and Asynchromix: Practical asynchronous MPC and its application to anonymous communication. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2019), pp. 887–903.

[158] Luu, L., Narayanan, V., Zheng, C., Baweja, K., Gilbert, S., and Saxena, P. A secure sharding protocol for open blockchains. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2016), pp. 17–30.

[159] Manuskin, A. The fastest draw on the blockchain (BZRX example). *Medium*. https://medium.com/@amanusk/the-fastest-draw-on-the-blockchain-bzrx-example-6bd19fabdbe1, 22 July 2020.

[160] Maram, D., Malvai, H., Zhang, F., Jean-Louis, N., Frolov, A., Kell, T., Lobban, T., Moy, C., Juels, A., and Miller, A. CanDID: Can-do decentralized identity with legacy compatibility, Sybil-resistance, and accountability. In *IEEE Symposium on Security and Privacy (SP)* (2021. To appear.).

[161] Maram, S. K. D., Zhang, F., Wang, L., Low, A., Zhang, Y., Juels, A., and Song, D. CHURP: Dynamic-committee proactive secret sharing. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2019), pp. 2369–2386.

[162] Marino, B., and Juels, A. Setting standards for altering and undoing smart contracts. In *Symposium on Rules and Rule Markup Languages for the Semantic Web* (2016), pp. 151–166.

[163] Marino, W. Smart-contract escape hatches: The Dao of the DAO. *Hacking, Distributed*. https://hackingdistributed.com/2016/06/22/smart-contract-escape-hatches, 22 June 2016.

[164] Matetic, S., Schneider, M., Miller, A., Juels, A., and Capkun, S. Delegatee: Brokered delegation using trusted execution environments. In *USENIX Security Symposium (USENIX Security)* (2018), pp. 1387–1403.

[165] McCorry, P., Hicks, A., and Meiklejohn, S. Smart contracts for bribing miners. In *Financial Cryptography and Data Security (FC)* (2018), pp. 3–18.

[166] McCorry, P., Hicks, A., and Meiklejohn, S. Smart contracts for bribing miners. In *Financial Cryptography (FC)* (2018).

[167] Metcalfe, B. Metcalfe's law after 40 years of ethernet. *Computer 46*, 12 (2013), 26–31.

[168] Micali, S. Algorand's smart contract architecture. *Algorand Blog*. https://www.algorand.com/resources/blog/algorand-smart-contract-architecture, 27 May 2020.

[169] Miller, A., Bentov, I., Bakshi, S., Kumaresan, R., and McCorry, P. Sprites and state channels: Payment networks that go faster than Lightning. In *Financial Cryptography and Data Security (FC)* (2019), pp. 508–526.

[170] Miller, A., Cai, Z., and Jha, S. Smart contracts and opportunities for formal methods. In *Symposium on Leveraging Applications of Formal Methods* (2018), pp. 280–299.

[171] Moos, M. Mining pool censorship could make Zcash 'mostly unusable'. *Cryptoslate*. https://cryptoslate.com/mining-pool-censorship-zcash-unusable, 7 June 2019.

[172] Moroz, D. J., Aronoff, D. J., Lovejoy, J., Narula, N., and Parkes, D. C. Double-spend counterattacks. In *Cryptoeconomic Systems (CES)* (2020).

[173] Möser, M., Eyal, I., and Sirer, E. G. Bitcoin covenants. In *Financial Cryptography and Data Security (FC)* (2016), pp. 126–141.

[174] Narayanan, A., and Clark, J. Bitcoin's academic pedigree. *Communications of the ACM 60*, 12 (2017), 36–45.

[175] Nayak, K., Fletcher, C. W., Ren, L., Chandran, N., Lokam, S. V., Shi, E., and Goyal, V. HOP: Hardware makes obfuscation practical. In *Networks and Distributed Security Systems (NDSS)* (2017).

[176] Nazarov, S., Shukla, P., Erwin, A., and Rajput, A. Bridging the governance gap: Interoperability for blockchain and legacy systems. World Economic Forum whitepaper. https://www.weforum.org/whitepapers/bridging-the-governance-gap-interoperability-for-blockchain-and-legacy-systems, Dec. 2020.

[177] Nisan, N., Tardos, E., Roughgarden, T., and Vazirani, V. *Algorithmic Game Theory*. Cambridge University Press, 2007.

[178] Nowak, M., and Sigmund, K. A strategy of win-stay, lose-shift that outperforms tit-for-tat in the prisoner's dilemma game. In *Nature* (1993), vol. 364, pp. 56–58.

[179] Osipovich, A. More exchanges add 'speed bumps,' defying high-frequency traders. *Wall Street Journal* (29 July 2019).

[180] Papadopoulos, D., Wessels, D., Huque, S., Naor, M., Včelák, J., Reyzin, L., and Goldberg, S. Making nsec5 practical for dnssec. *Cryptology ePrintArchive, Report 2017/099* (2017).

[181] Pass, R., and Shi, E. Fruitchains: A fair blockchain. In *ACM Symposium on Principles of Distributed Computing (PODC)* (2017), pp. 315–324.

[182] Pass, R., and Shi, E. Hybrid consensus: Efficient consensus in the permissionless model. In *International Symposium on Distributed Computing (DISC)* (2017), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[183] Pass, R., Shi, E., and Tramèr, F. Formal abstractions for attested execution secure processors. In *Advances in Cryptology (EUROCRYPT)* (2017), pp. 260–289.

[184] Perez, D., and Livshits, B. Broken metre: Attacking resource metering in EVM. *arXiv preprint arXiv:1909.07220* (2019).

[185] Peterson, J., Krug, J., Zoltu, M., Williams, A., and Alexander, S. Augur: a decentralized oracle and prediction market platform (v2.0). https://augur.net/whitepaper.pdf, 2019.

[186] Poon, J., and Buterin, V. Plasma: Scalable autonomous smart contracts (working draft). https://www.plasma.io/plasma.pdf, 11 Aug. 2017.

[187] Poon, J., and Dryja, T. The Bitcoin lightning network: Scalable off-chain instant payments. http://lightning.network/lightning-network-paper.pdf, 2016.

[188] Protocol Labs. Filecoin: A decentralized storage network. https://filecoin.io/filecoin.pdf, 19 July 2017.

[189] Qin, K., Zhou, L., Livshits, B., and Gervais, A. Attacking the DeFi ecosystem with flash loans for fun and profit. *arXiv preprint arXiv:2003.03810* (2020).

[190] Reiter, M. K., and Birman, K. P. How to securely replicate services. *ACM Trans. Program. Lang. Syst. 16*, 3 (May 1994), 986–1009.

[191] Ritzdorf, H., Wüst, K., Gervais, A., Felley, G., and Capkun, S. TLS-N: Non-repudiation over TLS enabling ubiquitous content signing. In *Networks and Distributed Security Systems (NDSS)* (2018).

[192] Rivest, R. L., Shamir, A., and Tauman, Y. How to leak a secret: Theory and applications of ring signatures. In *Theoretical Computer Science, Essays in Memory of Shimon Even*. Springer, 2006, pp. 164–186.

[193] Roberts, J. J. Exclusive: Coinbase buys Xapo custody for $55 million, eyes lending business. *Fortune* (15 Aug. 2019). https://fortune.com/2019/08/15/coinbase-xapo-bitcoin-custody.

[194] Roberts, J. J., and Rapp, N. Nearly 4 million Bitcoins lost forever, new study says. *Fortune* (25 Nov. 2017).

[195] Robinson, D., and Konstantopoulos, G. Ethereum is a dark forest. *Medium*. https://medium.com/@danrobinson/ethereum-is-a-dark-forest-ecc5f0505dff, 28 Aug. 2020.

[196] Rutter, K. R3 reports: If at first you don't succeed, try a decentralized KYC platform: Will blockchain technology give corporate KYC a second chance?, 22 July 2018.

[197] Saltzer, J. H., and Schroeder, M. D. The protection of information in computer systems. *Proceedings of the IEEE 63*, 9 (1975), 1278–1308.

[198] Schultz, D. A., Liskov, B., and Liskov, M. Mobile proactive secret sharing. In *ACM Symposium on Principles of Distributed Computing (PODC)* (2008), ACM, pp. 458–458.

[199] Shacham, H., and Waters, B. Compact proofs of retrievability. In *Advances in Cryptology (ASIACRYPT)* (2008), pp. 90–107.

[200] Sinclair, S. Ethereum Classic suffers second 51% attack in a week. *Coindesk* (6 Aug. 2020). https://www.coindesk.com/ethereum-classic-suffers-second-51-attack-in-a-week.

[201] StarkWare. Volition and the Emerging Data Availability spectrum. *Starkware Blog*. https://medium.com/starkware/volition-and-the-emerging-data-availability-spectrum-87e8bfa09bb, 14 June 2020.

[202] Stinson, D. R., and Strobl, R. Provably secure distributed Schnorr signatures and a (t, n)-threshold scheme for implicit certificates. In *Australasian Conference on Information Security and Privacy* (2001), pp. 417–434.

[203] Team Rocket. Snowflake to Avalanche: A novel metastable consensus protocol family for cryptocurrencies. https://ipfs.io/ipfs/QmUy4jh5mGNZvLkjies1RWM4YuvJh5o2FYopNPVYwrRVGV, 16 May 2018. [Online; accessed 30 Mar. 2021].

[204] Tezos. Proof-of-Stake in Tezos. https://tezos.gitlab.io/whitedoc/proof_of_stake.html, [Online; accessed 30 Mar. 2021].

[205] The LibraBFT Team. State machine replication in the Libra blockchain. https://developers.libra.org/docs/assets/papers/libra-consensus-state-machine-replication-in-the-libra-blockchain/2020-05-26.pdf, 1 May 2020.

[206] Tomescu, A., Abraham, I., Buterin, V., Drake, J., Feist, D., and Khovratovich, D. Aggregatable subvector commitments for stateless cryptocurrencies. *IACR Cryptol. ePrint Arch. 2020* (2020), 527.

[207] Tramèr, F., Zhang, F., Lin, H., Hubaux, J.-P., Juels, A., and Shi, E. Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge. In *IEEE European Symposium on Security and Privacy (EuroS&P)* (2017), pp. 19–34.

[208] Travers, G. All Synths are now powered by Chainlink decentralised oracles. *Synthetix Blog*. https://blog.synthetix.io/all-synths-are-now-powered-by-chainlink-decentralised-oracles, 1 Sept. 2020.

[209] U.S. Securities and Exchange Commission. Updated investor bulletin: Accredited investors. https://www.investor.gov/introduction-investing/general-resources/news-alerts/alerts-bulletins/investor-bulletins/updated-3, 21 Jan. 2019.

[210] U.S. Securities and Exchange Commission. SEC modernizes the accredited investor definition. *SEC Press Release*. https://www.sec.gov/news/press-release/2020-191, 26 Aug. 2020.

[211] Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T. F., Yarom, Y., and Strackx, R. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security Symposium (USENIX Security)* (2018), pp. 991–1008.

[212] van Schaik, S., Kwong, A., Genkin, D., and Yarom, Y. SGAxe: How SGX fails in practice. sgaxe.com, 2020.

[213] van Vuuren, G. J. AES Golang encryption performance benchmarks updated. *Medium*. https://medium.com/@gerritjvv/aes-golang-encryption-performance-benchmarks-updated-bcfa3555165b, 30 June 2019.

[214] Vickrey, W. Counterspeculation, auctions, and competitive sealed tenders. In *The Journal of Finance* (1961), vol. 17, pp. 8–37.

[215] Vorick, D., and Champine, L. Sia: Simple decentralized storage. https://sia.tech/sia.pdf, 2014.

[216] W3C. Decentralized identifiers (DIDs) v1.0: Core architecture, data model, and representations. *W3C Working Draft*. https://w3c-ccg.github.io/did-spec, 4 Feb. 2021.

[217] Walton-Pocock, T. Aztec: Fast privacy with ZK^2 rollup. *Aztec Blog*. https://medium.com/aztec-protocol/aztec-fast-privacy-with-zk^2-rollup-7c742f45457, 27 Mar. 2020.

[218] Wang, X., Malozemoff, A. J., and Katz, J. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, [Online; accessed 30 Mar. 2021].

[219] Wang, X., Ranellucci, S., and Katz, J. Global-scale secure multiparty computation. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2017), pp. 39–56.

[220] Weng, C., Yang, K., Katz, J., and Wang, X. Fast, scalable, and communication-efficient zero-knowledge proofs for boolean and arithmetic circuits. Cryptology ePrint Archive, Report 2020/925, 2020. https://eprint.iacr.org/2020/925.

[221] Wesolowski, B. Efficient verifiable delay functions. *Journal of Cryptology* (2020), 1–35.

[222] Whinfrey, C., Fontaine, S., Guido, D., Daian, P., and Breidenbach, L. Failure to set gasLimit appropriately enables abuse. https://drive.google.com/file/d/1mULop1LxHJJy_uzVBdc_xFItN9ck04Jj/view, 19 Nov. 2018.

[223] Wiggins, M. W. Vigilance decrement during a simulated general aviation flight. *Applied Cognitive Psychology 25*, 2 (2011), 229–235.

[224] Wikipedia contributors. Txt record — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/TXT_record, 2020. [Online; accessed 30 Mar. 2021].

[225] Wikipedia contributors. Accelerated mobile pages — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Accelerated_Mobile_Pages, 2021. [Online; accessed 30 Mar. 2021].

[226] Wilkinson, S. Storj: A peer-to-peer cloud storage network (v1.01). https://storj.io/storj2014.pdf, 15 Dec. 2014.

[227] Wilson, D., and Ateniese, G. From pretty good to great: Enhancing PGP using Bitcoin and the blockchain. In *International conference on network and system security (NSS)* (2015), pp. 368–375.

[228] Witkowski, J., and Parkes, D. C. Peer prediction without a common prior. In *ACM Conference on Electronic Commerce (EC)* (2012), pp. 964–981.

[229] Xing, B. C., Shanahan, M., and Leslie-Hurd, R. Intel® Software Guard Extensions (Intel® SGX) software support for dynamic memory allocation inside an enclave. In *Hardware and Architectural Support for Security and Privacy*. 2016, pp. 1–9.

[230] Yasskin, J. Signed HTTP Exchanges. *W3C Internet-Draft*. https://wicg.github.io/webpackage/draft-yasskin-http-origin-signed-responses.html, 27 Jan. 2021.

[231] Yin, M., Malkhi, D., Reiter, M. K., Gueta, G. G., and Abraham, I. Hotstuff: BFT consensus with linearity and responsiveness. In *ACM Symposium on Principles of Distributed Computing (PODC)* (2019), pp. 347–356.

[232] Zamani, M., Movahedi, M., and Raykova, M. Rapidchain: Scaling blockchain via full sharding. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2018), pp. 931–948.

[233] Zhang, F., Cecchetti, E., Croman, K., Juels, A., and Shi, E. Town Crier: An authenticated data feed for smart contracts. In *ACM Conference on Computer and Communications Security (ACM CCS)* (2016), pp. 270–282.

[234] Zhang, F., Maram, S. K. D., Malvai, H., Goldfeder, S., and Juels, A. DECO: Liberating web data using decentralized oracles for TLS. In *ACM Conference on Computer and Communications Security* (2020), pp. 1919–1938.

[235] Zhang, X.-Z., Liu, J.-J., and Xu, Z.-W. Tencent and Facebook data validate Metcalfe's law. *Journal of Computer Science and Technology 30*, 2 (2015), 246–251.

[236] Zhang, Y., Setty, S., Chen, Q., Zhou, L., and Alvisi, L. Byzantine ordered consensus without Byzantine oligarchy. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)* (2020).

[237] Zhou, L., Qin, K., Torres, C. F., Le, D. V., and Gervais, A. High-frequency trading on decentralized on-chain exchanges. *arXiv preprint arXiv:2009.14021* (2020).

[238] Zimmermann, P. R. *The official PGP user's guide*. MIT Press Cambridge, 1995.

A Glossary

This glossary is intended to define important terms in the body of the paper. Square brackets denote the section of the main body of the paper (if any) in which a given term receives treatment. Boldface terms within definitions indicate a corresponding glossary entry.

- **Authenticated data origination (ADO)** [Section 7.1]**:** The use of digital signatures by original data sources to authenticate data fed from their APIs. By authenticating data provenance, ADO can help strengthen the integrity of oracle reports.

- **Byzantine Fault Tolerance (BFT):** A term used of distributed systems protocols such as blockchains. BFT is the property of ensuring security even if a (minority) fraction of nodes / servers behave in a Byzantine, i.e., arbitrarily malicious, manner. The term Byzantine also conveniently describes the mechanics of most Byzantine-Fault Tolerant protocols.

- **Cryptoeconomics:** A domain-specific form of **mechanism** that leverages cryptographic techniques and digital assets to create desirable equilibria within decentralized systems.

- **DECO** [Section 3.6.2]**:** Short for *decentralized oracle*, DECO is a new cryptographic protocol that enables a user (or oracle) to prove statements in zero-knowledge about data obtained from HTTPS-enabled servers. DECO consequently allows private data from *unmodified* web servers to be relayed safely by oracle networks. (It *does not* allow data to be sent by a prover directly on chain.) DECO has narrower capabilities than Town Crier, but unlike Town Crier, does not rely on a trusted execution environment.

- **Exchange:** A platform for digital-asset trading. *Decentralized Exchanges* (DEXes) are exchanges realized in part or full as smart contracts.

- **Fair Sequencing Services (FSS)** [Section 5]**:** Chainlink's functionality to bring blockchain users the innovation of first-come, first-serve services like the kind you can get at a deli. FSS helps prevent front-running and other schemes that arbitrageurs use to gain a technical advantage over ordinary users in existing smart contract systems.

- **Front-running:** A profit-making strategy in financial systems such as public blockchains in which a trader, observing an incoming transaction, exploits it by speedily placing her own transaction first. The term has a narrower meaning on Wall Street than in the blockchain community, but an even more thriving community of practitioners.

- **Future fee opportunity:** The net revenue a Chainlink oracle may expect to receive from service payments from customers in the future. Loss of such fees in cases of malfeasance represents an opportunity cost.

- **Game theory:** Mathematically formal study of strategic interaction. A game is a model of such an interaction, typically in the real world, that codifies sets of actions available to participants in the game, known as *players*.

- **Layer-2 systems:** A class of approaches to improving the performance— both transaction throughput and latency—of blockchains. Layer-2 systems involve *off-chain* processing of transactions, followed by periodic reconciliation / syncing with the target blockchain. **Rollups** are one example.
- **Mechanism / mechanism design (MD):** Sometimes called "inverse **game-theory**," MD is the science of designing games, called mechanisms, that incentivize particular behaviors by participants, conventionally known as *players*.
- **Public-key infrastructure (PKI):** A system for secure creation and management of a mapping of identities to public keys.
- **Rollup** [Section 6.3]**:** A variant on the classical computer science transaction-processing approach known as batching, now available in fruit flavors. Rollups can help increase transaction processing speeds and throughputs, and are a promising approach to scaling blockchains to handle high transaction loads. A rollup involves off-chain construction of batches of transactions, followed by on-chain submission. Two popular rollup design approaches are:
  - *Optimistic rollups:* A class of rollup schemes in which transaction batches are submitted on chain without verification. Their correctness is enforced by allowing participants to mount challenges to incorrect rollups, and using staking as an economic incentive for such challenges.
  - *zk-Rollups:* A class of rollup schemes in which 'zk' inexplicably denotes "zero-knowledge," a property that zk-Rollups need not and often do not have. zk-Rollups use succinct arguments of knowledge, i.e., compact cryptographic proofs or correct computation, to enable efficient verification when submitted on chain.
- **Super-linear / quadratic staking impact** [Section 9.4.2]**:** An adversary can corrupt rational, i.e., incentive-driven, oracle nodes by *bribing* them. Given an adversary with budget $\$B(n)$, and n participating oracle nodes, each with a fixed staking deposit $\$d$, we say that a staking scheme has *super-linear impact* if adversarial success requires $B(n)$ that grows asymptotically faster than n. *Quadratic impact* means that $B(n)$ grows asymptotically as n^2.
- **Staking** [Section 9]**:** A means of reinforcing the security of a decentralized system using rigorously quantifiable economic incentives. In a staking scheme, participants provide a cryptocurrency deposit that is partially or fully forfeited in cases of participant deviation from a blockchain protocol. In the Chainlink setting, oracle nodes perform staking to ensure correct reports; they forfeit stake upon failure to produce a correct report.

- **Transaction-Execution Framework (TEF)** [Section 6]**:** A Decentralized Oracle Network framework for layer-2 systems that implements oracle support and Fair Sequencing Services.
- **Town Crier** [Section 3.6.2]**:** An oracle technology that uses a trusted execution environment (TEE) to enforce integrity and confidentiality on oracle data and computation. Town Crier has capabilities that subsume those of DECO, but requires trust in a TEE.
- **Trusted Execution Environment (TEE):** An execution environment for applications, often with special-purpose hardware support. A TEE aims to provide strong security properties such as confidentiality and application integrity, i.e., tamper-resistance for software. TEEs are a work in progress, but one that promises to have a large industry impact.
- **Trust minimization** [Section 7]**:** A common term in the blockchain industry cleverly crafted to mean the opposite of what it sounds like it should mean. Trust minimization refers to a desirable design feature in decentralized systems: Strong assurance of correct behavior under limited assumptions about the trustworthiness of system components.

B DON Interface: Further Details

In this appendix, we provide more details on the DON interface outlined in Section 3. We treat the three computing resources of networking, computation, and storage respectively in Appendix B.1, B.2, and B.3.

B.1 Networking

Oracle systems have arisen to address a limitation in basic blockchain protocols: public / permissionless blockchains cannot securely fetch data from external systems, such as web servers.[20] DONs aim to support application-level adapters between blockchains and off-chain systems in a general, flexible, and extensible form.

In this section, we discuss the techniques DONs will use to construct secure adapters and to ensure their availability. We also give some examples of their application. Most of the techniques we describe have not been codified before in a Chainlink whitepaper. They can be applied in Chainlink as it is architected today, i.e., do not require DONs, and we expect that some of them will be.

[20]Most data of interest is not digitally signed by the originating web server. Thus data fetched by a single miner producing a block would be vulnerable to tampering. Even in committee-based consensus protocols, e.g., [103, 120], there would be formidable challenges in realizing oracle functionality, including data availability failures (and their potential impact on performance and/or transaction censorship), added low-level complexity, access restrictions on some data sources (e.g., proprietary sources), etc.

We characterize the security of an adapter in terms of the classic CIA (confidentiality, integrity, availability) triad.

Of greatest importance in an oracle system is data *integrity*. Integrity here means that an adversary cannot tamper with data relayed from the source to the DON (or vice versa). In some circumstances, a channel should also provide *confidentiality*. Confidentiality means that an adversary learns only the data that a protocol is designed to send over the channel. (We provide more precise notions.) A channel should exhibit *availability*, meaning that the channel is created and carries data in a timely way in response to relying parties' needs. Finally, as an addition to the triad, we note that *accountability* is important to ensure the proper functioning of oracle systems: In cases of failure, it should be possible to identify (and penalize) malfunctioning nodes.

B.1.1 Integrity

Transport Layer Security (TLS)—the dominant internet protocol for establishing a secure channel between a client and a host—has a serious inherent limitation: It provides integrity *only for the two communicating parties*. It does not digitally sign data in a way that supports proof to a third party that data originated with the host (or client), and proposals for this purpose, e.g., [9, 191], are currently seeing very limited adoption. Some data of interest for blockchain systems may be digitally signed by its producers, and we discuss approaches to support this potentially beneficial practice in Section 7.1. Today, however, the vast majority of web data isn't digitally signed.

DONs therefore incorporate purpose-built mechanisms for ensuring the integrity of a report r communicated *to* the DON from one or more other sources. These come in three forms, each with its own associated trust and operational model.

To describe the functioning of the three mechanisms, we consider a simple model in which players aim to ensure the integrity of what we call an *enhanced* report $r = (\tilde{r}, \mathsf{sid}, S)$, where a basic report payload $\tilde{r}$ is tagged with: (1) sid, a session identifier and (2) an identifier S for the source / entity with which the report originated. The value sid provides a unique label for the protocol context in which $\tilde{r}$ is sent, and may include such information as a unique identifier of the relying smart contract, an unique identifier of the query to which the report corresponds, etc. We refer to an enhanced report simply as a *report* where clear from context.

We denote by $\rho_i = (r\,[, w])$ a report that an oracle node $\mathcal{O}_i$ writes to $\mathcal{L}$, where w denotes an (optional) *witness*. We refer to w (with or without a witness) as a *recorded report*. Because players' messages on $\mathcal{L}$ are authenticated, i.e., signed by the players that post them, writing ρ_i can serve as an endorsement by $\mathcal{O}_i$ in a relying protocol.

DONs support three basic mechanisms for ensuring the integrity of queries and reports, meaning their being relayed from claimed source S without tampering. They are:

- **Cryptographic proofs:** Digital signing of r by a source S with public key pk, i.e., $\rho = (r, w = \mathsf{Sig}_{\mathsf{pk}}[r])$, provides a direct assurance of the integrity of the message.

- **Trusted execution environments (TEEs) / Town Crier:** An application instance A with public key pk_A running in a fully featured TEE, such as Intel SGX, can generate an *attestation*, that is, a digitally signed witness $w = \mathsf{Sig}_{\mathsf{pk}_A}[r]$. Assuming trust in the TEE, then, the recorded report $\rho_A = (r, w = \mathsf{Sig}_{\mathsf{pk}_A}[r])$ ensures the integrity of r.

- **Thresholding:** Any subset of $f + 1$ players generating recorded reports $\rho_i = (r, \mathsf{Sig}_{\mathsf{pk}_i}[r])$ or combined or aggregate witness $\rho_{\mathcal{O}} = (r, \mathsf{Sig}_{\mathcal{O}}[r])$ provides a collective view of r as originating correctly with source S.

Thresholding and TEEs are complementary. They can be used simultaneously for a stronger combined security assurance, in the sense that integrity is assured if at least one of the two mechanisms is sound.

We may extend our definition of integrity to encompass recording a transformation $\mathsf{conf}(\tilde{r})$ of $\tilde{r}$, rather than $\tilde{r}$ itself. This extension is useful for confidentiality, as discussed below.

Untrustworthy sources: A given data source S may, of course, provide untrustworthy data. Adapter integrity in the sense we have defined it does not include mechanisms to address corrupt sources. The predominant mechanism used in oracle systems is aggregation over sources. For example, if an asset's price reports r_1, r_2, r_3 are fetched from sources S_1, S_2, S_3, an aggregate report $r = \mathsf{median}(r_1, r_2, r_3)$, given one faulty source, r will be the value reported by a correct source. Ensuring correct aggregation is a problem of computation integrity, as discussed in Appendix B.2.

B.1.2 Confidentiality

We say that an adapter has *weak confidentiality* if $\mathcal{O}$ learns both the query q and report data r generated by the adapter. Weakly confidential protocols may still provide other forms of confidentiality, however: the adapter may place only partial information about (q, r) on $\mathcal{L}$ and thus partially conceal (q, r) from an observer, and auxiliary data in the protocol may still be confidential.

We say an adapter has *strong confidentiality*, on the other hand, if q and r are partially or fully concealed from $\mathcal{O}$. It's not obvious at first glance how to achieve strong confidentiality: How, after all, can $\mathcal{O}$ process a hidden query and manage a hidden report?

DONs can use DECO [234] and/or Town Crier [233] to achieve strong confidentiality in adapters. See Appendix C for an example.

DECO and Town Crier are designed to provide legacy compatibility, i.e., the ability to use existing data sources, in a confidentiality-preserving way. An alternative approach is to change the output of data sources by digitally signing it and providing confidentiality features, as in TLS-N [191], a nicely complementary approach that we will also explore, and to a limited extent in OpenID Connect [9], an identity-oriented data-signing protocol that runs on top of OAuth 2.0.

B.1.3 Availability

The availability of an adapter depends on both the availability of the system to which it is connecting as well as the availability of the DON nodes responsible for implementing the adapter. While availability mechanisms may be adapter-specific, we expect that they will commonly rely on a few basic techniques already used or slated for inclusion in the current version of Chainlink. One, of course, is *redundancy*, i.e., having multiple nodes operate the channel. Another is the use of cryptoeconomic incentives to reinforce fast channel operation; for example, while redundant nodes receive payment for their services, a premium may be paid for fast execution, e.g., a reward for the first node to relay data to the DON from an off-chain system.

See Appendix B.1 for further details on adapters in DONs.

B.2 Computation

In their support of layer-2 scaling approaches—whether based on optimistic computing or validity proofs—DONs offer the possibility of significant performance improvements for smart contracts. By employing additional technologies, DONs can provide further performance enhancement and also support various forms of *confidentiality*, whether for oracle data as an adjunct to privacy-preserving layer-2 schemes (e.g., [217]), or to provide privacy enhancement for schemes in which it isn't present.

Zero-knowledge proof (argument) systems, such as zk-SNARKs [126], zk-STARKs [38], and Bulletproofs [56, 66], are powerful tools for privacy and will play an important role in the evolution of confidentiality-preserving smart contract systems [149], as they do, for instance, in DECO. It is worth noting that the goal of proving statements in zero-knowledge to an oracle node admits use of a new form of zero-knowledge proof with a *designated verifier* (i.e., a proof that is verifiable only by a particular entity, not publicly) [220]; such schemes can greatly outperform more general-purpose zero-knowledge proof systems. We expect that zero-knowledge proofs will play an important role in trustworthy and confidential relaying of oracle results, in both adapters and executables. But they have a critical limitation: They *don't conceal secrets from the prover*.

To enable more general confidentiality-preserving functionality will require additional techniques. These same techniques can also enforce *integrity*, i.e., help ensure that computation is performed correctly. We briefly discuss the roles of *Trusted Execution Environments (TEEs)* and *Secure Multi-party Computation (MPC)* in the evolution of Chainlink.

B.2.1 Trusted Execution Environments (TEEs)

TEEs such as Intel SGX [229, 133] and Keystone [152] allow applications to execute within a protected environment, known as an enclave, that in principle protects both their integrity and confidentiality. TEEs are designed to realize a powerful and very general functionality [183], one that encompasses even powerful cryptographic

primitives such as virtual black-box obfuscation, that are provably unrealizable using cryptography alone [27, 175].

The idea of leveraging TEEs for smart contracts was introduced in [139, 233] and several general-purpose frameworks for this purpose have since emerged [58, 80, 99].

We expect that TEEs will eventually play a role in the Chainlink network, as they are the basis for Town Crier [233], and Chainlink already supports TEE-based blockchain systems, e.g., [8]. As the technology continues to mature, we believe it will play an increasingly important role in blockchain systems in general, and Chainlink in particular.

Concerns about TEE security have been one impediment to their adoption. We therefore briefly discuss this issue.

B.2.2 TEE Security

Recent attacks against Intel SGX, the only commodity TEE with attestation capabilities, e.g., [211, 212], have cast doubt on the security of TEEs in practice. We believe that the technology will evolve, particularly with open source efforts emerging [152].

Mitigations: It is possible to deploy TEEs in ways that recognize their potential for compromise. One is to employ them for defense-in-depth, that is, for hardening systems. Town Crier may be viewed in this light: Trustworthy oracle operators should provide their users with confidentiality and integrity, but use of Town Crier strengthens this assurance. Similarly, TEEs can be deployed in ways that don't rely on perfect security. Sealed-glass proofs [207], for instance, use TEEs for integrity, but not confidentiality (although recent side-channel attacks compromise both).

Finally, it is possible to design TEE-based protocols with *forward security*, such that if a break becomes known: (1) Players can avoid use of the TEE and (2) The break does not retroactively impact previous protocol invocations. This approach would greatly limit the window of opportunity for adversaries to exploit TEE vulnerabilities. For example, in protocols where a TEE controls users' funds, users might move those funds into addresses specifically for temporary, TEE-only use. In protocols where TEEs enforce confidentiality for long-lived data, frequent key rotation would achieve a similar goal.

For example, a TEE could be used to implement the $\mathsf{ConfSwitch}$ adapter for confidential DeFi discussed in Appendix C.3. The TEE ingests (q, switch) encrypted under a public key $\mathsf{pk}_{TEE}(t)$ for an epoch t lasting, e.g., a week, and generates the output $s = \mathsf{switch}(r)$. It then purges all the data it has processed. At the beginning of each new epoch, it deletes its old key, generates a new one, and publishes it to $\mathcal{L}$ or $\mathsf{MAINCHAIN}$. (Such publication could be facilitated by an executable on $\mathcal{L}$ with a per-epoch initiator.) Players can additionally abort by agreement at any time.

Of course, users have exposure in this model to zero-day, i.e., unpublished, attacks. There is a cost to using a zero-day attack, however, namely the risk of its exposure and invalidation through publication of a patch.

B.2.3 Secure Multi-Party Computation (MPC)

MPC is a term for generic computation on secret-shared values by a set of nodes. MPC enables realization of a "black box" functionality, i.e., computation of a function g on confidential inputs $x_1, \ldots, x_z$ such that players learn only $g(x_1, \ldots, x_z)$, and learn no information about the inputs to the function or intermediate computations. Inputs can be secret-shared, and also represented in a binding but confidential way on chain using a commitment scheme or encryption under a secret-shared key.

While quite flexible, and despite impressive recent advances, *generic* MPC—i.e., MPC that supports any function g—incurs very high computation and communication overhead by comparison with direct execution on a single node. For example, MPC using fairly state-of-the-art tools operating in an adversarial model with malicious players—such as EMP toolkit [218, 219]—require about 20 seconds of computation for a single AES evaluation for 14 nodes on a WAN. (Only 250ms of computation is online; the rest can be performed in a precomputation phase.) By comparison, an AES evaluation on a modern, consumer-grade CPU requires around 2μs [213].

In short, generic MPC is reasonably practical, but only for modest computations. Practical implementations have tended to focus on relatively modest computational tasks, such as auctions in Hyperledger Fabric [42].

Considerably more efficient protocols can be realized for certain functions g. One example is threshold signing, which is especially practical for BLS signatures [54]. 2PC, computation involving just two players can leverage special techniques ("garbled circuits") that makes it considerably more efficient than computation involving $n > 2$ players. DECO [234], for example, uses both custom and generic 2PC techniques.

Most MPC schemes and implementations have some important limitations. One is that a single player can abort the protocol and learn its output, while (anonymously) preventing other players from learning theirs'. This means a failure of *robustness* (ability to complete computation) but also *fairness* (in an MPC-specific sense), meaning that all players receive outputs or none does.

Schemes that allow identification of aborting players (ID-MPC), e.g., [37] are one approach to this problem. But MPC protocols also generally make the strong assumption of network synchrony, without which they may experience both confidentiality and integrity failures. A reasonably practical robust version of MPC that operates in asynchronous networks is presented in [157].

Another limitation of most MPC protocols is their lack of *public verifiability*. This isn't problematic in a model that assumes a quorum of honest players, but would be for the type of MAINCHAIN accountability discussed in Section 7.3 and Section 8.3. To ensure correct MPC outputs in the case of a failure in a majority of nodes requires use of publicly verifiable variants such as [37] with a public bulletin board—necessarily MAINCHAIN in a majority-dishonest model—and is thus not terribly practical.

In summary, we expect that DONs will support MPC for specialized computations, such as threshold signing and DECO, early in their development. We will track developments in the fast-moving field of MPC to determine where support for generic MPC

may be appropriate.

B.3 Storage

Data sent to blockchains can play a multifarious role. It may be consumed immediately by contracts. It can alternatively be placed on chain for fast future access by contracts or for purposes not related to contract execution, such as archival storage facilitating the audit of off-chain systems. The quantities of data required for these last two purposes greatly exceeds that consumed by contracts today, as well as the data-storage capacity of most permissionless blockchains.

We envision DONs accommodating two forms of storage. The first is *on-ledger* storage, achieved by inclusion within a DON's ledger. The second option, particularly attractive for bulk storage, is *off-ledger* storage.

Off-ledger storage may involve the use of off-chain systems that are innately decentralized, such as Filecoin [188], IPFS [40], Storj [226], or Sia [215]. Filecoin has a built-in protocol called *Proof of Replication* (PoReps) [74, 112, 111, 188, 199], designed to ensure storage of multiple copies of a file F (and prevent miners from gaming the consensus protocol). The other three can only achieve this property by having an end user encrypt F multiple times under different keys, creating distinct corresponding ciphertexts $\tilde{F}_1$, $\tilde{F}_2$, $\tilde{F}_3$. A DON can perform such encryption on behalf of a user, permitting proof of replication without native support in a target storage protocol.

Where compatible with users' desired trust models, it is also possible to use existing systems, such as conventional cloud storage providers. Replication across such services is possible for robustness [57] and complementary to and composable with decentralized storage services.

Confidentiality: DONs include the ability for $\mathcal{O}$ to store data encrypted under a secret-shared key. They can perform confidentiality-preserving computation over such data, as discussed above in Appendix B.2.

B.4 Resource Pricing

Accurate charging of users for resource use on blockchains remains a perennial challenge. Mispriced resources do not merely carry the risk of discouraging important applications within an ecosystem. They also constitute a security risk, as they can enable denial-of-service attacks [184] and more subtle attacks such as drainage of funds [222].

Resource pricing of executable logic in a DON is a somewhat easier problem, as DON nodes control resource usage and can apply whatever pricing strategies they deem appropriate. Possible approaches include market-driven pricing, gas metering à la Ethereum, which is supported in EVM and WASM [3], and language-driven approaches such as non-Turing-complete languages [131]—potentially along with bounded code sizes. Pricing of adapters, however, will certainly require application-specific pricing

strategies, as adapters access external resources, e.g., cloud storage, whose cost is DON-independent.

C Adapter Examples

In this appendix, we give a few examples of adapters, to illustrate both the wide variety that is desirable and achievable in practice and relevant notions of security and the techniques to enforce them.

We use the notation $\mathsf{Adapter}(\mathcal{P}, \mathcal{S})$ to denote an adapter that involves players in $\mathcal{P}$ and a set of data sources (or a single data source) $\mathcal{S}$. We let $\rightarrow T$ denote an input / output to system T where there's ambiguity. Otherwise inputs should be understood as read by an adapter from $\mathcal{L}$ and outputs as written to $\mathcal{L}$.

C.1 Oracle-Mediated Data-Source Access (MediatedReport)

In some cases, certain nodes may have access to a data source S that the full set of nodes $\mathcal{O}$ does not. For example, some nodes may have API keys or other credentials that other nodes in $\mathcal{O}$ lack. Nodes with privileged access to a data source S can *mediate* access on behalf of $\mathcal{O}$ using DECO (or Town Crier). Here we describe an adapter that does so with weak confidentiality.

Technical approach: Using DECO, it is possible for a particular $\mathcal{O}_i$ to access S using confidential credentials and prove to another node $\mathcal{O}_j$ that it correctly relayed a report r from S, *without revealing its credentials to* $\mathcal{O}_j$. This approach can be scaled of course to proofs across subsets of nodes.

Adapter $\mathsf{MediatedReport}(\mathcal{O}, S)$

Input: q

Output: $\langle \texttt{medreport} : (q, r);\ \mathcal{O}_i \rangle$

Security:

- *Availability:* $\mathcal{O}_i$ must be available when invoked by players in $\mathcal{O}$ that lack access to S.
- *Integrity:* r is correct under the cryptographic hardness assumptions of DECO and assuming at least k honest nodes in $\mathcal{O}$.

Example application: Suppose $\mathcal{O}_i$ submits a report r from data source S whose value is disputed by another node or nodes in $\mathcal{O}$. This might occur, for instance, as part of the dispute mechanism in a staking protocol.

The dispute might be arbitrated by invoking the full set of nodes in $\mathcal{O}$. Some nodes in $\mathcal{O}$, however, may lack access to S, which may be a specialized data source. Using DECO, it is possible for $\mathcal{O}_i$ to prove to nodes in $\mathcal{O}$ that its report r was correctly relayed from S, *without other nodes having direct access to* S.

Variants and extensions: Town Crier offers an alternative way to achieve the same adapter. The credentials of $\mathcal{O}_i$ can be placed in Town Crier instances run by individual oracle nodes in $\mathcal{O}$, with appropriate restrictions on their use [164].

It is also possible to extend MediatedReport so that $\mathcal{O}$ may call any node in $\mathcal{O}^*$, a subset of nodes with access to S. It is also possible to generalize MediatedReport to involve nodes on MAINCHAIN. Such a variant can be helpful in protocols, e.g., staking or optimistic rollups, e.g. [141], that use on-chain arbitration of off-chain computation.

C.2 Cross-Ledger Reports (XL-Report-Read)

For many applications, it can be useful to exchange data across DONs. Here we describe a simple adapter to import a basic report from a DON B to another DON A. For simplicity, we assume a unique report of the form $\langle \texttt{report} : (q, r) \rangle$ on $\mathcal{L}_B$ for any given q.

Technical approach: As messages on a ledger $\mathcal{L}_B$ are signed with respect to $\mathsf{pk}_{\mathcal{L}_B}$, they can easily be authenticated by the set of oracle nodes $\mathcal{O}_A$ operating Decentralized Oracle Network A if they know $\mathsf{pk}_{\mathcal{L}_B}$.

Adapter XL-Report-Read$(\mathcal{O}_A, \mathcal{L}_B)$

Input: $(q, \mathsf{pk}_{\mathcal{L}_B}) \to \mathcal{L}_A$, for query q and target DON public key $\mathsf{pk}_{\mathcal{L}_B}$

Output: $\langle \texttt{XLreport} : (q, r); \mathsf{pk}_{\mathcal{L}_B} \rangle \to \mathcal{L}_A$.

Security:

- *Availability:* At least one node in $\mathcal{O}_A$ must have access to $\mathcal{L}_B$.
- *Integrity:* r's correctness depends upon an honest $\mathcal{O}_B$.

Example application: A DON A may determine that it has sufficient trust in DON B to treat its reports as trustworthy. In this case, it may be more cost-effective to import certain reports than to generate them independently.

PKI support: The main challenge in securing report transfer between DONs is maintenance of a supporting *public-key infrastructure* (PKI), that is, a trustworthy,

up-to-date mapping of public keys to DONs. To support this essential functionality in Chainlink, we plan to maintain smart-contract-based directories of Decentralized Oracle Networks and their public keys on all blockchains that Chainlink supports.

Variants and extensions: Of course, many other possible cross-DON adapters are possible. For example, DON A may have permission in DON B to import *confidential* data from $\mathcal{L}_B$. In this case, the adapter would require participation from $\mathcal{O}_B$. As another example, DON A may have selective write permissions on DON B, in which case a cross-chain adapter for writing data would be possible.

Other extensions are possible to handle multiple or ambiguous reports on $\mathcal{L}_B$. For example, XL-Report-Read could relay the message $M = \big((q, *), z\big)$ with the highest index z on $\mathcal{L}_B$ at a given time t.

C.3 Confidential Switch (ConfSwitch)

Our previous two examples did not provide on-ledger confidentiality of q or r. We now describe one that does: ConfSwitch. This adapter reveals on chain only a function switch of an aggregate (medianized) report r, and is thus appropriate for price feeds. It provides on-ledger confidentiality for q and r, revealing only a function of the latter. It also enables recording of confidential data for an auditor (with public key $\mathsf{pk}_{\mathsf{aud}}$). Such auditing may be important, as the details of transactions that use ConfSwitch are not visible on the ledger or main chain. The data made available to the auditor includes q, switch, and optional auxiliary data α.

Technical approach: To provide confidentiality for q and switch (and optionally α), they are encrypted under the public keys of $\mathcal{O}$ and the auditor. $\mathcal{O}$ accumulates reports off ledger and aggregates them into a report r, and outputs $\mathsf{switch}(r)$. Neither q nor any report data are written explicitly on $\mathcal{L}$.

Strong confidentiality: Strong confidentiality is achievable in ConfSwitch using DECO or Town Crier.

In such a variant, players place on $\mathcal{L}$ the commitment $c = \mathsf{comm}(q, \mathsf{switch})$ to the query and switch. Either player can then use DECO or Town Crier to retrieve reports and prove in zero knowledge their correctness with respect to c. As DECO enables such proofs only for individual reports, an additional step would be required: The user would prove the correctness of ciphertexts on individual reports and then prove the correctness of a claimed output $\mathsf{switch}(r)$, i.e., correct medianization and application of switch to the reports.

Adapter $\mathsf{ConfSwitch}$ $(\mathcal{O}, \mathcal{S})$

Input: $\mathsf{Enc}_{\mathsf{pk}_{\mathcal{O}}}[q, \mathsf{switch}], \mathsf{Enc}_{\mathsf{pk}_{\mathsf{aud}}}[q, \mathsf{switch}, \alpha]$

Output: $\langle \texttt{ConfSwitch} : \mathsf{switch}(r) \rangle$, for $r = \mathsf{median}(\{r_i\}_{i:\mathcal{O}_i \in \mathcal{O}})$

Security:

- *Confidentiality:* The adapter reveals Mixicle execution details to $\mathcal{O}$, but only $\mathsf{switch}(r)$ to observers of $\mathcal{L}$ and $\mathsf{MAINCHAIN}$.
- *Availability:* Availability depends on full availability of $\mathcal{S}$ (although variants are possible that rely on a threshold subset of $\mathcal{S}$).
- *Integrity:* r's correctness depends upon cryptographic hardness assumptions, the honesty of $\mathcal{O}$, and the accuracy of reports from $\mathcal{S}$.

Figure 19: Adapter $\mathsf{ConfSwitch}$

D Functional Signatures

A functional (digital) signature scheme [59], like a conventional signature, has a master (private) key sk that permits signing of any message. Additionally, however, the master key can be used to create a signing key sk_F, for a function F, that permits signing of any value y in the range of F, i.e. such that $F(x) = y$ for some input x.

A simple way to realize this primitive, given in [59], is as follows (with slight modification). A fresh key pair $(\mathsf{sk}', \mathsf{pk}')$ is generated. To create the key sk_F for function F is computed as:

$$\mathsf{sk}_F = \Big(\mathsf{sk}', \mathsf{cert} = \mathsf{Sig}_{\mathsf{sk}}(\mathsf{pk}' \parallel F)\Big),$$

where cert here may be viewed as a "certificate" stating that pk' is a valid public key for a functional signature with function F and master public key pk.

A signature on message y using sk_F, then, takes the form:

$$\mathsf{Sig}^*_{\mathsf{sk}_F}(y) = (\mathsf{cert}, \mathsf{Sig}_{\mathsf{sk}'}(y), x).$$

To verify $\mathsf{Sig}^*_{\mathsf{sk}_F}(y)$, i.e., compute $\mathsf{verify}^*\Big(\mathsf{Sig}^*_{\mathsf{sk}_F}(y)\Big)$, a verifier checks that $F(x) = y$, and also checks the validity of cert and $\mathsf{Sig}_{\mathsf{sk}'}(y)$ in the obvious way.

The value x (the "witness") can in principle be large—as in the oracle-oriented use cases we consider. A straightforward remedy [59] that is computationally efficient for the verifier is to substitute for x a SNARK (or zk-SNARK for confidentiality)—a compact proof (argument) of the existence of x.

Functional signatures are conceptually straightforward, as are their basic constructions. But they provide a useful framework for reasoning about combining source data in oracle networks.

D.1 Functional Signatures for Combining Data

Returning to Example 4, we can see how a functional signature might be applied to create a proof σ that a value v was correctly computed from the value-signature set pair (V, Σ). Let $(\mathsf{sk}_i, \mathsf{pk}_i)$ denote the signing key pair for data source S_i in $\mathcal{S} = \{S_1, S_2, \ldots, S_{n_S}\}$. Suppose pk is a public key for $\mathcal{S}$ whose corresponding private key is shared in a threshold manner among data sources; for simplicity, assume sk_i is S_i's share of sk.

We can then define a functional signature that medianizes a set of n_S validly signed values as follows:

$$F(V, \Sigma) = \left\{ \begin{array}{ll} \mathsf{median}(V) & \text{if } \forall i \in \{1 \ldots n_S\}, \mathsf{verify}(\mathsf{pk}_i, \sigma_i, v_i) = \texttt{true} \\ \bot & \text{otherwise.} \end{array} \right\}$$

A functional signature scheme with master key sk, then, can be used to transmit a correctly computed median value v to SC. The oracle network computes $\sigma^* = \mathsf{Sig}^*_{\mathsf{sk}_F}(v)$, and sends it to SC, which verifies it by computing $\mathsf{verify}^*(\sigma^*, \mathsf{pk}, v)$.

Of course, to improve robustness to data source failures, F can be relaxed to medianize a subset of at least k_S validly signed values for some threshold $k_S \leq n_S$, i.e., to accept valid signatures from a quorum of sources, as we do in our example constructions below.

D.2 Discretized Functional Signatures

Computing σ^* using a SNARK-based function-signature construction is a computationally intensive task that may be incompatible with high performance DON applications. We therefore propose a relaxed variant we call *discretized functional signatures*, the key idea of which is to place a grid over the space of possible signed values, and constrain the values to that grid. For example if the required answer is a single scalar, the grid could be all the integer multiples of some grid size $a \in \mathbb{R}^+$, $a\mathbb{Z} = \{\ldots, -3a, -2a, -a, 0, a, 2a, 3a, \ldots\}$. Then a valid signed value is a representation of some az for $z \in \mathbb{Z}$ plus a valid signature over az.

The value az need not be the precise answer $F(V, \Sigma)$, but will be close to it. To formalize this, in addition to the predicate function F from Appendix D.1, the specification of a discretized functional signature has an additional pair of parameters (δ, δ'), corresponding to how coarse/fine the constraining grid is. They are as follows:

- *The parameter* δ bounds permitted deviation from the correct output of F. Informally,[21] given a pair (V, Σ), we require that it be infeasible for an adversarial signer (the DON, for our purposes) to generate a signature σ^* that deviates from the correct output of $F(V, \Sigma)$ by more than δ, i.e., such that:

$$\mathsf{verify}^*(\sigma^*, \mathsf{pk}, v) \bigwedge (|F(V, \Sigma) - v| > \delta) .$$

- *The parameter* δ' characterizes how precisely some quorum of values in V must match, for a feasible signature to be guaranteed. I.e., a functional signature σ^* can always be constructed, given k_S signers $\{S_{z_i}\}_{i=1}^{k_S}$ with observations $\{v_{z_i}\}$ such that $|v_{z_i} - v_{z_j}| \leq \delta'$ for all i, j.

Thus, δ may loosely be viewed as a *soundness* parameter, while δ' is a *completeness* parameter, in the sense that it specifies that signatures can always be constructed from observations $\mathbf{v} = (v_1, \ldots, v_{k_S})$ by honest participants as long as $\mathbf{v}$ lies within δ' of the diagonal in V^{k_S}-space, under the max norm, i.e. so the larger δ' is, the larger the set of signable observation sets is.

Recall our assumption that sk_i is a secret share of sk. In describing an example construction, we further assume that Sig is a non-interactive threshold signature, e.g., a BLS signature [54]. In an example below, we will use (k_S, n_S)-threshold signatures for some threshold $k_S \leq n_S$ such that $k_S > \frac{2}{3} n_S$.

With this machinery, we can build efficient functional signatures for a variety of different message spaces. The idea, briefly, is for data sources collectively—but non-interactively—to compute partial (k_S, n_S)-threshold signatures on grid values $az_i \in a\mathbb{Z}$ in the neighborhood of their respective individual values v_i. In this way, the data sources in $\mathcal{S}$ collectively constrain v, without explicitly computing it. If at least k_S data sources emit values v_i that are sufficiently close, their neighboring grid values will have a common element which they have all signed, collectively creating a threshold signature on a grid point within a δ-bounded neighborhood of a valid $v = F(V, \Sigma)$. A signature $\mathsf{Sig}^*_{\mathsf{sk}}(v)$ for $v = az$ is simply a threshold signature on $az \in a\mathbb{Z}$.

We illustrate by way of example, describing a construction for medianizing. It is convenient here to define a *cell* for $a\mathbb{Z}$ as an interval of values contained within a pair of successive grid points, i.e., $[az, a(z + 1)]$.

Example 6 (Discretized functional signature for median)**.** *Suppose the value to be signed is a price quote (e.g., the USD price of ETH), and can thus be represented as a positive integer in* $\mathbb{Z}^+$ *(e.g., denominated in some fraction of US dollars such as cents). For the function* $F(V, \Sigma)$ *that medianizes a set of at least* k_S *signed source values, the following construction achieves* $(\delta = a, \delta' = a)$ *for any* $a \in \mathbb{N}$.

Let

[21] A more formal security definition would require application of the "discretized" property to the existential unforgeability under a chosen message attack (EUF-CMA) for threshold signatures, for which see, e.g. [118].

$$\begin{aligned}\iota(v_i) &= a\lfloor v_i/a \rfloor, \\ \iota'(v_i) &= \iota(v_i) + a,\end{aligned}$$

i.e., $[\iota(v_i), \iota'(v_i)]$ *is the highest cell in the grid* $a\mathbb{Z}$ *which contains* v_i*.*

Each data source S_i *emits a signature* σ_i *of the following form:*

$$\sigma_i = (\sigma_i^{(1)}, \sigma_i^{(2)}, \sigma^{(3)}) = \left(\mathsf{Sig}_{\mathsf{sk}_i}\left(\iota(v_i)\right), \mathsf{Sig}_{\mathsf{sk}_i}\left(\iota(v_i)\right), \mathsf{Sig}_{\mathsf{sk}_i}\left(v_i\right)\right)$$

A signature σ_i *is valid if all the constituent signatures* $\sigma_i^{(j)}$ *are valid, and the messages in* $\sigma_i^{(1)}, \sigma_i^{(2)}$ *are equal to* $\iota(v_i), \iota'(v_i)$*, where* v_i *is the message in* $\sigma_i^{(3)}$*.*[22]

A functional signature $\mathsf{Sig}^*_{\mathsf{sk}}(v) = F(\{v_i\}, \{\sigma_i\})$ *can be constructed from the signatures* $\{\sigma_i\}_i$ *if there is a common grid point* $v = az \in a\mathbb{Z}$ *on which some sufficiently large subset of* $\{\sigma_i^{(1)}, \sigma_i^{(2)}\}_i$ *contains partial signatures.*

The scheme in Example 6 achieves $\delta = a$ and $\delta' = a$ for the median function $F(V, \Sigma)$ that medianizes a set of at least k_S signed source values, as we will show in the following lemmas.

Lemma 1. *Given a valid discretized functional signature on* $v \in a\mathbb{Z}$*, constructed from* k_S *partial signatures on* v*,*

$$\left\{\sigma_i = (\mathsf{Sig}_{\mathsf{sk}_i}(\iota(v_i)), \mathsf{Sig}_{\mathsf{sk}_i}(\iota'(v_i)), \mathsf{Sig}_{\mathsf{sk}_i}(v_i))\right\}_{i=1}^{k_S},$$

it follows that each v_i *lies in* $[v - a, v + a]$*.*

Proof. Since $\{\sigma_i\}$ can be used to construct a functional signature on v, we must have $v \in \{\iota(v_i), \iota'(v_i)\}$ for each i. Thus one of the following is true:

$$\begin{aligned} v = \iota(v_i) &= a\lfloor v_i/a \rfloor &&\implies v_i \in [v, v + a] \\ v = \iota'(v_i) &= a\lfloor v_i/a \rfloor + a &&\implies v_i \in [v - a, v] \end{aligned}$$

Therefore, $v_i \in [v - a, v] \cup [v, v + a] = [v - a, v + a]$. □

[22] $\sigma_i^{(3)} = \mathsf{Sig}_{\mathsf{sk}_i}(v_i)$ is not necessary for the correctness of this construction, but including it avoids distracting questions about what the median of the observations is, when

- a dishonest oracle S_i could attest to arbitrary grid cell boundaries irrespective of its observed value v_i, or even without having observed any value v_i at all, or
- an aggregator of the partial signatures could choose an arbitrary k_S-sized subset of the σ_i's from which to construct the functional signature

It is not hard to optimize this construction to work with only $\Sigma = \{(\sigma_i^{(1)}, \sigma_i^{(2)})\}$, and show that the range of possible values for the median implied by the resulting functionally-signed value covers all possible actual median values, as long as less than $\frac{n_S}{3}$ answers are dishonest. It requires the minor (but distracting) ontological gymnastics of reassigning inaccurately-reported/nonsensical/nonexistent values of V to arbitrary integers.

Lemma 2. *Let $\{v_i\}_{i=1}^m$ be a set of observations, and let $I \subset \{1, \ldots, m\}$ be a subset of indices which constitute a majority. The median of $\{v_i\}$ lies in $[\min(\{v_i\}_{i\in I}), \max(\{v_i\}_{i\in I})]$.*

Proof. Without loss of generality, assume the observations $v_1, \ldots, v_m$ are sorted. The subsequence

$$\arg\min_{i\in I} v_i, \ldots, \arg\max_{i\in I} v_i$$

is longer than $\frac{m}{2}$, since $|I| > \frac{m}{2}$. Therefore, the midpoint(s)[23] of the sequence $1, \ldots, m$ lie(s) in subsequence, and thus the median lies in $[\min(\{v_i\}_{i\in I}), \max(\{v_i\}_{i\in I})]$. □

Lemma 3. *Given a valid discretized functional signature on $v \in a\mathbb{Z}$, from a set of n_S signers $\{S_i\}$, more than two thirds of which are honest, and a signature threshold $k_S > \frac{2}{3}n_S$, the true median lies in $[v - a, v + a]$.*

Proof. By lemma 1, all the v_i's used by honest signers lie in $[v - a, v + a]$. While dishonest signers could have signed v on the basis of any observation (or even no observation), the honest participants form a majority of the signers, so by lemma 2 the median observation lies between two honest values in $[v - a, v + a]$, and therefore itself lies in $[v - a, v + a]$. (The honest signers form a majority because at least $k_S > \frac{2}{3}n_S$ signed, but the number of dishonest participants in the signing group is assumed to be less than $\frac{1}{3}n_S = \frac{1}{2} \times \frac{2}{3}n_S < \frac{k_S}{2}$.) □

Lemma 4. *The discretized functional signature scheme in Example 6 achieves $\delta = a$ for the medianization function.*

Proof. By lemma 3, the true median lies in $[v - a, v + a]$. All points in this interval are less than a units from v, so the distance from v to the true median is less than a. I.e., the condition from the definition of δ which we desire to be infeasible,

$$\mathsf{verify}^*(\sigma^*, \mathsf{pk}, v) \bigwedge (|F(V, \Sigma) - v| > \delta)\,.$$

is indeed infeasible, for $\delta = a$. □

Lemma 5. *The discretized functional signature scheme in Example 6 achieves $\delta' = a$. I.e., given a quorum of at least k_S signers $Q = \{S_i\}$ with observations $|v_i - v_j| < a$ for $S_i, S_j \in Q$, a valid discretized functional signature can be constructed from their signatures $\{\sigma_i = (\mathsf{Sig}_{\mathsf{sk}_i}(\iota(v_i)), \mathsf{Sig}_{\mathsf{sk}_i}(\iota'(v_i)), \mathsf{Sig}_{\mathsf{sk}_i}(v_i))\}$*

Proof. Let $I = [\min(\{v_i\}_i), \max(\{v_i\}_i)]$ be the smallest interval containing $\{v_i\}_i$. Since the length of I, $|\max(\{v_i\}_i) - \min(\{v_i\}_i)|$, is a difference between two elements of $\{v_i\}$ it is less than a. Thus I contains at most one point of $a\mathbb{Z}$. If it contains one, say v, then $v \in \{\iota(v_i), \iota'(v_i)\}$ for all I, because each v_i lies in either the cell below or the cell above v. Therefore a functional signature on v can be constructed from the σ_i's. If it does not, it lies entirely within one cell of $a\mathbb{Z}$, and $\{\iota(v_i), \iota'(v_i)\}$ will be the same for all i, so a functional signature can be constructed on either boundary value. □

[23] There are two midpoints if m is even, $\frac{m}{2}$ and $\frac{m}{2} + 1$, and one midpoint if m is odd, $\frac{m}{2} + 1$.

The technique in Example 6 can be viewed as a means of reducing proximity testing to equality testing by discretizing values within a lattice. Greater resilience to variance (in the sense of larger δ') can be achieved if participants send signatures over a larger set of grid points in the neighborhood of their observed v_i's.

Enriching the discretized functional signature: As described here, a signature $\mathsf{Sig}^*_{\mathsf{sk}}(v)$ is on a grid point $v \in a\mathbb{Z}$, and v will be close to but not necessary equal to the median v_{med} computed over a set of at least k_S node-supplied values. It is possible, however, for a DON *also* to include v_{med} in its signed message. If v_{med} is computed by the DON, then it is a trustworthy value *assuming the DON is functioning correctly*. In some cases, contracts may wish to consume v_{med} if $|v - v_{\mathsf{med}}| \leq a$, use v as assurance against gross inaccuracy of v_{med}.

E Prospective Bribery

One class of adversarial behavior we expressly designed our proposed staking mechanisms to protect against is what we call *prospective bribery*. This novel attack is a generalization of the bribery attacks described in [166]. Prospective bribes are feasible in and can impact the security of proof-of-work, proof-of-stake, and permissioned systems.

In proof-of-work and proof-of-stake blockchains, a prospective briber commits to paying the creator of a future block a specified bribe given that the future block meets some specified condition. For example, a briber could post a bribe to ensure that a future block does *not* include a specific transaction, or includes a specified *ordering*.

We use the term *prospective* because the briber actually does not know which node will be chosen to lead the protocol in the relevant round. Still, the bribe can be successful as long as the candidates for that leadership role are aware of the bribe.

To grasp the impact of prospective bribery in the context of staking, consider a staking mechanism for an oracle network in which there is a pool of n oracles, and in each round a node is selected at random—using fresh randomness—to unilaterally report the next value. It may appear that this strategy is naturally resistant to bribery attack, as a briber might have no way to determine ahead of time which node to bribe. However a briber can offer a guaranteed bribe to whichever node is selected as leader of the target block, promising a payment if and only if the block returns `false` (to use the language of Section 9).

A briber can guarantee payment of a prospective bribe using a smart contract analogous to those shown in [166].

F Random vs. Committee-Based Oracle Selection

It is possible to operate an oracle network in a permissionless or quasi-permissionless way, selecting a random subcommittee $\tilde{\mathcal{O}} \subset \mathcal{O}$ producing a given report r or sequence of reports. An ostensible benefit of this approach is its resistance to attacks that corrupt nodes, whether technical or bribery-based. Provided that $\tilde{\mathcal{O}}$ isn't known in advance, an adversary must in principle control close to a majority of $\mathcal{O}$, rather than just $\tilde{\mathcal{O}}$, in order to subvert r with high probability for sufficiently large n.

This observation underlies many hybrid consensus protocols, e.g., [120, 182] as well as some oracle protocols, e.g., [25].

We believe, however, that in many applications, it will be beneficial for relying parties to *select stable oracle nodes (or committees)*, for several reasons:

- *Non-uniform corruption:* Rational nodes' reliability is a function of incentives that reflect factors beyond their staked assets, such as potential future revenue, reputation, and so forth, as discussed in Section 9. Users may prefer to rely on nodes they assess as being highly trustworthy, rather than randomly selected nodes. Even as compared with stake-weighted randomized selection, this choice can result in selection of a more trustworthy committee, i.e., one less likely to be corrupted.

- *Corruption thresholds:* A related issue is the fact that unless n and k are quite large or the fraction of adversarial nodes in $\mathcal{O}$ is quite low, the probability of a randomly selected subset $\tilde{\mathcal{O}}$ having a high proportion of corrupted nodes *at some time* can be quite high.

  For example, suppose that $n = 100$, $f = 10$, i.e., only 10% of nodes are corrupted, and a subcommittee of size $|\tilde{\mathcal{O}}| = 10$ is selected uniformly at random. The probability of a majority corrupt subcommittee being selected is ≈ 0.000009. Even in a system with low throughput, handling, e.g., one request a second, if a new committee is selected for each job, a corrupt committee will surface on expectation in *just over a day*.

  For this reason, given adversarial fraction $f/n = 1/4$, with its proposed security parameters, the random-committee-selection scheme in [41] requires a committee size of 26,091!

  We consequently believe that random selection is not sufficient to ensure the trustworthiness of $\mathcal{O}$ over time, an issue that is particularly pertinent given the next point we highlight.

- *Confidentiality:* In order to access data stored with oracle nodes in a threshold manner, a user must interact with the nodes that hold $\mathsf{sk}_{\mathcal{L}}$. It is possible to hand off $\mathsf{sk}_{\mathcal{L}}$ across committees of oracle nodes with dynamically changing membership. Apart from the potentially large increase in communication cost, frequent changes could be problematic because they result in an elevated probability of

the committee having $f \geq k$ corruptions *at some time*, resulting in a catastrophic failure, as explained above.

- *Prospective bribery:* As noted in Section 9, an adversary can offer a bribe to nodes in a committee *in advance* of committee selection by making the bribe *contingent* on a node's selection. That is, a node can prove that it was selected and generated a desired report in order to claim a bounty. Smart contracts can in fact facilitate such malfeasance using techniques like those in [165].

- *Innovation:* Oracle providers may choose to differentiate themselves based on performance, accuracy, or special features, such as enhanced confidentiality or access to proprietary data. Allowing users to choose providers, rather than drawing providers from a pool with homogeneous capabilities, can help promote innovation.

- *Customization:* Different users may have different preferences in terms of tradeoffs among reliability, performance, and cost tradeoffs, and should be able to express these preferences in terms of their selection of providers.