A Taxonomy and Security Analysis of Rollup Architectures in Ethereum Blockchain

Adrià TORRALBA-AGELL*, Muoi TRAN†, Cristina PÉREZ-SOLÀ‡
*Universitat Oberta de Catalunya, *atorralbaag@uoc.edu*
†Chalmers University of Technology and University of Gothenburg, *muoi@chalmers.se*
‡Universitat Autònoma de Barcelona, *cristina.perez@uab.cat*

***Abstract*—Layer 2 rollups improve Ethereum's scalability by executing transactions off-chain while relying on on-chain verification for finality. Despite their growing importance with increasingly high stakes, rollups have received limited systematic security analysis. In this paper, we present a taxonomy of rollup architectures and examine their associated security threats. We identify three main architectural patterns, primarily determined by the design of the sequencer, a critical component that bridges off-chain transaction execution with on-chain finality. Our analysis also reveals inherent trade-offs between throughput, decentralization, and resilience, demonstrating that no single architecture simultaneously optimizes all these properties.**

Index Terms— Network Attacks, Rollups, Sequencer Architectures, Layer 2 Scaling, Denial-of-Service

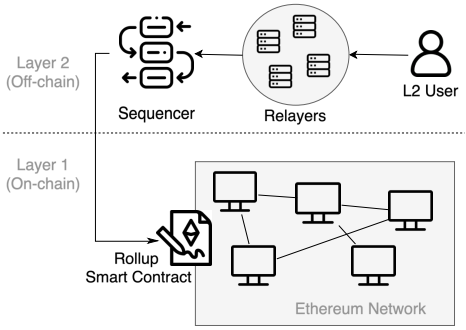

Figure 1. The transaction workflow in Ethereum rollups.

I. Introduction

Layer 2 (L2) scaling solutions have achieved significant adoption on the Ethereum blockchain, securing over $40 billion in total value locked [1] and processing more than 85% of Ethereum ecosystem transactions [2]. Among many L2 solutions, *rollups* have emerged as the primary paradigm, enabling near-instant transaction finality and high throughput. Rollup solutions are often grouped based on their security guarantee mechanisms, namely, optimistic rollups with interactive dispute workflow on Layer 1 (L1) [3] and ZK-rollups with validating zero-knowledge proofs [4]. At the core of any rollup solution lies a critical component called *sequencer* that orders and executes the user-submitted L2 transactions before batching results and posting them to L1.

With the extremely high stakes involved, L2 systems naturally exhibit security risks, several of which have already materialized in real-world incidents. In December 2023, the Arbitrum One sequencer experienced a critical two-hour outage when a software bug, exacerbated by high inscription-driven throughput, caused the system to stall [5]. Also, in June 2024, the Linea zkEVM rollup unilaterally paused its sequencer to censor attacker addresses following a $6.8 million exploit at the Velocore DEX [6], [7]. More recently, a Starknet sequencer failure in September 2025 caused a 3-hour outage and even forced a block reorganization [8]. Collectively, these incidents underscore the need for a deeper understanding of weak points in such emerging L2 solutions.

However, there exist many rollup implementations — L2Beat lists 24 different solutions, and this list is constantly growing [1]. Analyzing each of the solutions for their security risks is obviously not scalable. Moreover, to the best of our knowledge, there exists no comprehensive framework that identifies the threat models of rollup solutions in Ethereum.

This paper contends that all rollup solutions adhere to a limited set of recurring architectural patterns built from shared core components within their transaction workflows. To support this claim, we present a systematic taxonomy of Ethereum rollup architectures, focusing on their sequencer designs (§III). We also find that because these architectures share common components, rollups are exposed to similar attack surfaces and require comparable threat models (§IV). We conclude that no existing rollup design simultaneously satisfies the demands of throughput, decentralization, and resilience, and we outline several directions for future research (§V).

II. Background

A. Motivation for Rollups

Ethereum derives strong security guarantees from global consensus over the platform's state, which imposes a fundamental scalability limitation on transaction throughput captured by the blockchain trilemma [9]. To overcome this bottleneck without compromising security and decentralization, Ethereum has adopted a rollup-centric roadmap [10]. In this model, Ethereum L1 acts primarily as a secure settlement and data-availability layer, while L2 systems handle scalable transaction execution. Rollups accomplish this by moving execution off-chain, batching results back to L1 for verification, and relying on cryptographic proofs to guarantee the correctness of state transitions [11].

B. Identities and Roles in Rollups

Despite considerable diversity in their deployment models, Ethereum rollups share a common high-level architecture composed of four key entities, see Figure 1.

At the edge of the system are *users*, who submit transactions through publicly accessible RPC endpoints. From a user perspective, the desired properties mirror those of Ethereum itself: timely confirmation, deterministic execution, and eventual finality on L1. While users interact solely through wallet interfaces, their experience depends heavily on the downstream infrastructure responsible for confirming their transactions.

Sitting between users and the rollup protocol is a layer of *relayers* (i.e., public or semi-public RPC endpoints) that ingest user transactions and forward them to the sequencer. These relayers exhibit substantial variation in operational redundancy, geographic deployment, and trust assumptions, all of which directly influence transaction latency, censorship resistance, and privacy. In current rollup deployments, most public RPC relayers are operated either by third-party infrastructure providers (e.g., Alchemy [12], Infura [13]) or by the rollup teams themselves. They are typically easy to discover and are often openly listed in public registries [14].

The *sequencer* is the core operational component of any rollup. It can be a single operator, a set of redundant operators, or (in emerging systems) a distributed protocol. The sequencer is responsible for ordering incoming transactions, executing them against the L2 state machine, producing updated state roots, and periodically batching transaction data for submission to Ethereum L1. By maintaining a private mempool with full visibility into pending transactions, the sequencer determines the rollup's throughput, latency, and fairness properties.

Finally, each rollup includes a *verifier* contract deployed on Ethereum L1. The verifier acts as the authoritative on-chain arbiter of state correctness. It checks rollup state transitions using one of two verification models: fraud proofs (in optimistic rollups) or validity proofs (in ZK rollups). Regardless of the mechanism, the verifier enforces the rollup's security guarantees by accepting only state roots that can be cryptographically justified, thereby anchoring all off-chain activity to Ethereum's base consensus rules.

C. Transaction Workflow in Rollups

A user transaction in a rollup system progresses through three distinct stages, spanning off-chain execution, on-chain commitment, and eventual verification on L1 (Figure 1).

Stage 1: off-chain execution. The workflow begins when the sequencer receives a user transaction via RPC relayers. The sequencer orders the transaction deterministically relative to other pending transactions and executes it against the current L2 state machine, producing an updated state root. From the user's perspective, this yields a *soft confirmation*, i.e., rapid acknowledgment that the transaction has been accepted into the sequencer's private mempool. Importantly, soft confirmations rely on sequencer honesty, as until the results are posted and verified on L1, they remain provisional and subject to reordering or censorship.

Stage 2: on-chain commitment and data availability. Executed transactions are then aggregated into batches and posted to Ethereum L1 as either calldata [15] or blob data (EIP-4844 [16]), along with the sequencer's claimed post-state root. This posting provides *data availability*, the property that all transaction data necessary to reconstruct the rollup's state is made publicly accessible. Note that data availability enables credible neutrality, meaning anyone can run a full rollup node alongside a full Ethereum node by retrieving posted transaction data and re-executing state transitions, thereby verifying the state independently of the sequencer [17]. Also, data availability ensures forced withdrawal safety because even if the sequencer halts or becomes malicious, users can construct Merkle proofs of fund ownership and withdraw assets to L1, provided historical transaction data remains available on-chain [17]. Moreover, data availability serves as a foundational security property: if data were unavailable, users could neither independently verify the rollup's state nor withdraw funds in the case of a non-cooperative sequencer. Once a batch is posted, it enters a verification window on L1, whose duration and semantics depend on the underlying verification model.

Stage 3: verification and finality. The final stage occurs on Ethereum L1, where the verifier contract checks the sequencer's state transition claims using one of two following fundamentally different mechanisms. First, in optimistic rollups, commitments are accepted tentatively and placed into a challenge window (e.g., 7–14 days). During this period, any observer may present a fraud proof demonstrating that the sequencer produced an incorrect state transition. A successful challenge results in rejection of the batch, slashing of the sequencer's bond, and a reward to the challenger. If no valid challenge arises, the batch is finalized at the end of the window. Second, in the cryptographic proof model, ZK rollups accompany each batch with a succinct validity proof (e.g., a zk-SNARK [18] or zk-STARK [19]) attesting to the correctness of all included state transitions. The verifier contract checks the proof deterministically, and upon acceptance, finalizes the batch immediately without any delay or dispute period. Both workflows preserve the essential data availability invariant. Particularly, all transaction data is retained on L1, allowing independent state reconstruction and ensuring that users retain a safe withdrawal path even in the presence of sequencer failure or malicious behavior [20].

III. Rollup Infrastructures

We classify rollup systems based on the architecture of their most critical component, namely the sequencer. Indeed, although rollups vary widely in design goals and implementation details, their sequencers fall into three broad architectural patterns that meaningfully shape security, performance, and failure modes, i.e., stand-alone (§III-A), pipeline (§III-B), and distributed (§III-C). We also show all studied rollups and their corresponding architectures in Table I.

Stand-alone	Stand-alone (HA)	Pipeline	Distributed
Polygon zkEVM	Arbitrum One	Linea	Lighter
ZKSync Era	Base	Starknet	INTMAX
ZKSync Lite	Optimism	Katana	Facet Bluebird
Scroll	Unichain	Paradex	ZKBase
Zk.Money v1	Ink	Abstract	Starknet Grinta
Loopring	BOB	ZERO	
Zircuit			
Phala			
Space and Time			

Table I
EXISTING ROLLUPS [1] AND THEIR ARCHITECTURES.

A. Stand-alone Sequencer (with High Availability)

The most common and historically earliest deployment model for rollups is the stand-alone sequencer architecture, in which a single trusted operator is responsible for orchestrating the full lifecycle of L2 transactions. This design appears in two variants, namely simple and high-availability, reflecting different trade-offs between operational simplicity and resilience.

Simple stand-alone sequencer. The simplest instantiation of a rollup employs a single, centralized sequencer with direct RPC exposure and no internal redundancy. In this setup, the sequencer exclusively handles transaction ingestion, ordering, and execution, while also maintaining a persistent connection to an Ethereum node for posting transaction batches and corresponding state commitments to L1.

This simple architecture prioritizes minimal operational complexity, predictable behavior, and the lowest possible latency by avoiding any form of consensus or inter-operator coordination. However, these systems inherit a significant limitation: the sequencer represents a single point of failure. If it becomes unreachable (e.g., due to hardware failure, software bugs, DDoS attacks, or network partition), L2 production halts immediately. During downtime, users typically cannot force transaction inclusion or initiate withdrawals on L1 until the sequencer recovers. To partially mitigate this risk, some systems implement an escape hatch mechanism [20], which allows users to bypass the sequencer and submit certain critical transactions directly to Ethereum. Still, the recovery experience remains slow and disruptive, and the architecture offers limited resilience against targeted failures.

High-Availability (HA) stand-alone sequencer. To address these concerns while retaining the performance benefits of a centralized sequencer, some rollups deploy a high-availability variant of this architecture. Here, multiple sequencer replicas run in parallel, each deterministically executing the same transaction ordering logic and maintaining synchronized state. A dedicated availability coordinator designates one replica as the active sequencer responsible for ingesting user transactions, producing L2 blocks, and publishing L1 commitments, while the non-active replicas continuously stay in sync to enable a seamless failover if the active instance becomes unavailable. Figure 2a provides an overview of this design. To the user, this appears as a single RPC interface, which is typically fronted by a CDN or load balancer that routes incoming transactions to the active sequencer.

However, this redundancy introduces new operational risks absent in the simple model. Misconfigurations in the load balancer may route transactions to stale or incorrect replicas; compromised replicas or coordinators could collude to submit invalid state commitments; and the coordinator itself becomes a critical dependency whose failure can prevent replica selection or block production. Thus, while the high-availability architecture strengthens liveness, it broadens the trusted computing base and creates new attack surfaces that must be carefully managed.

B. Pipeline Sequencer

Pipeline sequencer architectures adopt a multi-stage design that decomposes the functionality of a monolithic sequencer into a set of specialized components. In this model, the end-to-end sequencing and proving workflow is organized as a pipeline consisting of the following stages. First, the sequencing node is responsible for transaction ordering and for producing a deterministic execution trace. A dedicated trace-generation module then executes the ordered transactions and outputs the corresponding trace, which may be optionally compressed by a Conflator. The resulting artifact is forwarded to a Coordinator, which schedules proving tasks to one or more provers, potentially arranged in a distributed or recursive configuration, to generate the final validity proof submitted to Ethereum. Figure 2b provides an overview of this workflow.

A key advantage of this design is that it decouples transaction latency from proof latency. Since the sequencer can publish a new state root immediately after completing execution, the system can offer near-instantaneous soft confirmations. Nevertheless, this architecture continues to exhibit network-level single points of failure, and it even worsens the situation since, instead of a single point of failure, this architecture presents several single point of failure components. Indeed, although functionality is decomposed, each L2 transaction must traverse every stage in the pipeline. Consequently, delays or failures in any intermediate component can propagate to downstream stages, creating bottlenecks and potentially causing cascading system-wide slowdowns.

C. Distributed Sequencer

The most recent generation of rollup designs explores distributed sequencer architectures, which replace the traditional single-operator sequencer (or its high-availability replication) with a set of independent sequencers that collectively order transactions. In these systems, sequencers are selected and coordinated through decentralized mechanisms, most commonly Byzantine fault-tolerant (BFT) consensus protocols [21], rotating-leader Proof-of-Stake (PoS) schemes [22], or auction-based selection. Rather than relying on a monolithic trusted operator, a committee of n sequencers participates in a protocol (e.g., Tendermint [23], PBFT [21]) to agree on transaction ordering, produce the canonical L2 block, and finalize the corresponding state transition. Figure 2c illustrates this architecture.

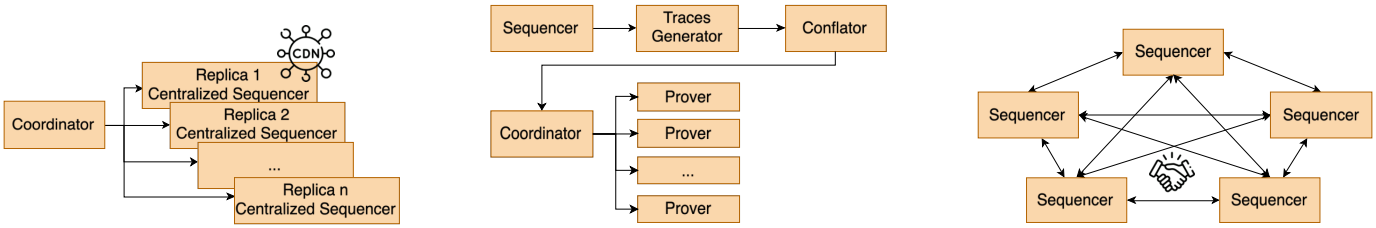

Figure 2. Three architectural designs of sequencers: (a) stand-alone (with high availability), (b) pipeline, (c) distributed.

The conceptual advantages of this design are substantial. First, censorship resistance is improved because as long as a threshold of sequencers (typically $\frac{2n}{3}$) remains honest, no single entity can unilaterally suppress or reorder user transactions. Second, fault tolerance and liveness become more robust, as progress depends on the collective behavior of a distributed committee rather than the availability of a single operator.

However, the design also introduces notable technical and economic challenges. Coordinating independent sequencers requires continuous consensus messaging, increasing communication overhead, and reducing throughput relative to centralized or pipeline-based designs. The multi-party environment also amplifies MEV-related risks, as competing sequencers may attempt to reorder transactions for profit. Systems must therefore adopt mitigations such as randomized leader election, encrypted mempools, or commit-reveal schemes. Furthermore, BFT-style consensus protocols remain vulnerable to liveness attacks, in which adversarial sequencers delay or obstruct the agreement process, temporarily halting L2 block production. Finally, the effectiveness of these architectures depends critically on incentive, e.g., robust mechanisms are needed to reward honest participation, deter equivocation or censorship, and enable slashing or exclusion of misbehaving sequencers.

IV. A Security Analysis of Rollups

Following our description of the rollup transaction workflow and its underlying architectural designs, we now turn to a systematic security analysis of rollup systems. In particular, we delineate the attack surfaces associated with each phase of transaction processing, namely before (§IV-A), during (§IV-B), and after (§IV-C) sequencer execution. For each attack surface, we examine concrete vulnerabilities and illustrate how both known and unexplored vectors may be leveraged.

A. Before Transaction Execution

Before user transactions reach the sequencer, they rely on off-chain pathways, primarily RPC relayers and Internet routing infrastructure, that constitute the pre-submission attack surface. We identify three key risks in this stage. First, public RPC relayers are the dominant interface through which users submit transactions, making them susceptible to a range of availability attacks, including volumetric DDoS, connection exhaustion, and application-layer abuse. Misconfigurations or software flaws in RPC gateways may further expose them to crashes or resource exhaustion. Because the sequencer depends on continuous transaction inflow, outages at the RPC layer can effectively halt user interaction with the rollup, even when the L2 backend remains fully operational.

Second, RPC operators can selectively censor or deprioritize transactions before they reach the sequencer. Such censorship may be explicit, motivated by regulatory constraints, address blacklisting, or compliance policies, or implicit, driven by MEV incentives. For example, an RPC provider could filter or delay transactions that conflict with the operator's own arbitrage or liquidation strategies. Since this filtering occurs upstream of the sequencer's mempool, it leaves no on-chain evidence, making such censorship difficult for users to detect.

Third, the communication path between users and RPC relayers traverses multiple network domains and is vulnerable to routing-layer and traffic-level attacks. Adversaries may perform BGP hijacks [24], route leaks, or targeted packet dropping to isolate certain users or regions selectively. State-level firewalls, ISP throttling, and local network interference can further disrupt or bias transaction propagation. These network-layer manipulations can systematically degrade access to the rollup or impose location-dependent censorship without compromising the relayer itself.

B. During Transaction Execution

Sequencer nodes constitute the most critical attack surface in any rollup system, as they are solely responsible for transaction ordering, execution, and producing state commitments. A compromised or stalled sequencer directly threatens liveness and, in some cases, short-term censorship resistance. The vulnerability profile differs significantly across architectural patterns (i.e., (HA) stand-alone, pipeline, and distributed), with increasing complexity generally widening the attack surface. We analyze the key risks across these architectural classes.

Attacks against stand-alone sequencers. In the simple stand-alone architecture, the sequencer represents a critical single point of failure. Key attack vectors include direct flooding, where adversaries overwhelm the sequencer with bursts of L2 transactions or malformed requests, saturating CPU, memory, or network resources. Because block production is tightly coupled to real-time execution, such resource exhaustion immediately halts L2 progress, preventing users from submitting transactions or initiating withdrawals until the operator manually restores service. Attackers may also submit

pathological or computationally expensive transactions (e.g., deeply nested smart contract executions) to exhaust sequencer validation resources. This depletes capacity, delaying or blocking legitimate transactions, and is particularly effective in systems with complex EVM semantics or limited transaction admission controls. Additionally, stand-alone sequencers must track L1 state, including forced transaction inclusions and escape hatch withdrawals [20]. By flooding Ethereum L1 with forced-inclusion transactions, adversaries can compel the sequencer to ingest, verify, and integrate these requests, delaying normal L2 processing and potentially degrading liveness.

HA stand-alone sequencer architectures replace a single sequencer node with multiple replicas coordinated through failover or leader-election mechanisms. While this design mitigates obvious single-node outages, it introduces additional attack surfaces targeting coordination, replication fidelity, and routing infrastructure. For instance, attackers may craft transaction sequences that exploit nondeterministic execution paths, race conditions, or timing variations across replicas. Even minor divergences in state roots or execution traces can trigger failover events or force replicas into resynchronization cycles, reducing throughput or stalling block production. The coordinator, which is often implemented via Redis, specialized databases, or custom leader-election services, represents another critical vulnerability. It can be targeted through connection exhaustion, state corruption, or network isolation. If the coordinator stalls, failover cannot occur, leaving the entire cluster without an active sequencer. Moreover, HA architectures rely on load balancers to distribute traffic evenly across replicas. Adversaries can exploit routing rules, improperly sanitized request headers, or edge-case behaviors to direct disproportionate traffic to weaker replicas. Such manipulations can overload certain nodes or induce state divergence, destabilizing the entire cluster.

Attacks against pipeline sequencers. Pipeline sequencer designs decompose transaction sequencing into multiple distributed components, including execution trace generators, conflators, batch coordinators, and prover networks. This modularization improves scalability and decouples execution from proof latency, but it also introduces inter-stage dependencies that attackers can exploit. Because pipeline sequencers rely on strict sequencing across interdependent services, failures in any stage can propagate downstream, amplifying the impact of attacks compared to monolithic designs. In particular, attackers can craft transactions with maximal computational complexity, placing disproportionate load on the trace generation component. Downstream components, such as conflators, coordinators, and provers, cannot proceed until trace outputs are available, causing delays that propagate through the entire pipeline and degrade overall system throughput. Network-layer attacks (e.g., BGP hijacks, targeted network interference) can also isolate critical pipeline components from either upstream sequencers or downstream provers. Such isolation halts batch formation and proof assignment, creating cascading delays that persist even when individual sequencer components remain operational. Misconfigured routing, load balancers, or internal APIs can further exacerbate these vulnerabilities by desynchronizing stages or overloading specific nodes. Moreover, pipeline architectures rely on distributed proving networks to generate cryptographic proofs for L1 verification. By submitting transactions that produce highly complex batches, attackers can saturate prover resources, delaying proof generation and finalization. In extreme cases, unproven batches may accumulate faster than the network can process, threatening system stability and extending L1 finality times.

Attacks against distributed sequencers. In distributed sequencer architectures, multiple independent nodes collectively determine transaction ordering and state transitions using consensus or sequencer selection protocols. While decentralization improves censorship resistance and reduces reliance on any single node, it introduces new attack surfaces targeting liveness, coordination, and economic incentives. For example, distributed sequencers typically rely on Byzantine fault-tolerant protocols (e.g., Tendermint [23], PBFT [21]) to agree on transaction ordering. Adversaries can delay, withhold, or reorder consensus messages to trigger repeated timeout rounds, slowing or stalling block production. Network-layer attacks can isolate subsets of nodes, preventing the formation of the required honest-majority quorum and effectively freezing the rollup until connectivity is restored. These attacks exploit the interdependence of nodes and amplify the impact of minor failures. If auction-based or PoS rotation mechanisms are used, attackers may acquire a disproportionate likelihood of being selected as sequencers. This enables targeted censorship of transactions or addresses and facilitates MEV extraction. Repeated influence over block proposals can compromise fairness and economic integrity, even when most sequencers act honestly. Also, some distributed designs rely on auxiliary services for leader election, task assignment, or state synchronization. These centralized or semi-centralized components can be targeted through connection exhaustion, state corruption, or misconfiguration. Compromising the coordinator stalls failover procedures and requires manual intervention, undermining the robustness benefits of a distributed architecture.

C. After Transaction Execution

Once the sequencer has executed transactions and produced a new state root, the rollup must post batch data to Ethereum L1 to guarantee data availability and enable independent verification. This data is published either as calldata or as blob space introduced in EIP-4844, forming the canonical record required for fraud proofs, validity proofs, and forced withdrawals [17]. At this stage, the attack surface shifts from L2-specific infrastructure to Ethereum's shared resource environment, which is outside the direct control of rollup operators. Although these attacks do not compromise correctness, they can substantially delay user withdrawals, extend finality times, and degrade the perceived responsiveness of the rollup. We highlight the following principal threats.

First, for rollups that rely on EIP-4844 blob storage, an adversary may flood the blob market with low-value or maliciously crafted blobs, consuming available capacity. This con-

gestion increases blob fees, delays data posting, and indirectly slows finality on the rollup. Since blob space is a shared global resource, attackers do not need to target a specific rollup to cause degradation effects.

Second, in the pipeline sequencer architecture where batch posting is delegated to a specialized component distinct from the transaction-executing sequencer, the batch poster becomes an additional point of failure. Targeted DoS attacks, resource exhaustion, or network isolation of the batch poster can prevent the timely submission of commitments to Ethereum, even when the sequencer operates correctly. Such decoupling improves scalability but enlarges the attack surface.

Third, for rollups using calldata, attackers can exploit Ethereum L1 fee dynamics by artificially inflating gas demand. By increasing calldata usage across unrelated applications or by submitting gas-intensive transactions, an adversary can drive up posting costs. Rollup operators may delay or batch more aggressively to reduce expenses, thereby increasing confirmation latency and extending the verification window for users awaiting finality or withdrawal.

V. Conclusion and Future Work

This work provides a systematic taxonomy of Ethereum Layer 2 rollups and analyzes the attack surfaces exposed throughout the transaction workflow. Our study demonstrates that architectural choices strongly influence both security and availability. Simple stand-alone sequencers offer operational simplicity and high throughput but are susceptible to single points of failure, resource exhaustion, and censorship. High-availability and pipeline designs mitigate some of these vulnerabilities, yet introduce new risks such as coordination failures, pipeline bottlenecks, and inter-component dependencies. Fully distributed sequencer architectures enhance censorship resistance and resilience under honest-majority assumptions but face practical challenges related to consensus liveness, network partitioning, and economic manipulation. Across all designs, the trade-offs between throughput, decentralization, and resilience are evident, highlighting that no single architecture optimally satisfies all requirements.

Several directions remain for advancing Ethereum rollup solutions that balance security, scalability, and decentralization. Formal verification of sequencer protocols and failover mechanisms could provide provable safety and liveness guarantees under adversarial conditions. Scalable Byzantine fault-tolerant consensus for distributed sequencers can reduce centralization risks while maintaining performance. Enhancing data availability, through redundancy, sampling, or tighter L1 integration, is critical for ensuring finality under network disruptions or DoS attacks. Protocol-level safeguards to limit MEV and prevent transaction manipulation are needed to preserve fairness and economic integrity. Finally, as Layer 2 ecosystems evolve, extending threat modeling to multi-rollup and cross-rollup interactions will be essential to secure interoperability and systemic resilience.

VI. Acknowledgments

This work was supported by the projects PID2021-125962OB-C33 SECURING/NET, PID2021-125962OB-C31 SECURING/CYBER, PID2024-156914OB-C43 SAFE-BLOCKCHAIN, and PID2024-156914OB-C41 SAFE-CYBER, funded by the Ministerio de Ciencia e Innovación, la Agencia Estatal de Investigación, and the European Regional Development Fund (ERDF); the ARTEMISA International Chair of Cybersecurity (C057/23) and the DANGER Strategic Project of Cybersecurity (C062/23), funded by the Spanish National Institute of Cybersecurity via the European Union – NextGenerationEU and the Recovery, Transformation and Resilience Plan; and the Catalan AGAUR grants SGR2021-00643 and SGR2021-01508. Additionally, A. Torralba-Agell is supported by AGAUR grant 2023 FI-100241 and a doctoral grant from the UOC.

References

[1] L2BEAT, "The state of the layer two ecosystem," https://l2beat.com/scaling/summary.
[2] Z. Spirkovski, "Layer 2s Now Handle 85% of All Transactions, Dune Report Says," 2025.
[3] Ethereum, "Optimistic Rollups," https://ethereum.org/developers/docs/scaling/optimistic-rollups/.
[4] ——, "Zero-knowledge rollups," https://ethereum.org/developers/docs/scaling/zk-rollups/.
[5] N. Grech, "Arbitrum Sequencer Outage | Root Cause Analysis," https://dedaub.com/blog/arbitrum-sequencer-outage/.
[6] Linea.eth, "Update on Velocore Incident," https://x.com/LineaBuild/status/1797283402745573837, 2024.
[7] ForkLog, "Velocore DEX Suffers $6.8 Million Loss in Exploit," https://forklog.com/en/velocore-dex-suffers-6-8-million-loss-in-exploit/, 2024.
[8] Z. Vardai, "Ethereum L2 Starknet suffers 2nd mainnet outage in 2 months," https://cointelegraph.com/news/starknet-outage-ethereum-l2-reliability-concerns, 2025.
[9] A. Torralba-Agell and C. Pérez-Solà, "A Comparison of Layer 2 Techniques for Scaling Blockchains," in *Proc. RECSI*, 2022.
[10] M. Neuder and A. Stokes, "Rollup-Centric Roadmap," https://notes.ethereum.org/@mikeneuder/rcr2vmsvftv, 2025.
[11] A. Torralba-Agell and C. Pérez-Solà, "Security Assumptions in Permissionless Blockchains and Layer 2 Scalability Solutions: A Comprehensive Taxonomy," *Blockchain: Research and Applications*, 2025.
[12] "Alchemy," https://www.alchemy.com/.
[13] "Infura," https://www.infura.io/.
[14] "ChainList," https://chainlist.org/.
[15] Ethereum, "Blockchain Data Storage Strategies," https://ethereum.org/developers/docs/data-availability/blockchain-data-storage-strategies/.
[16] V. Buterin, D. Feist, D. Loerakker, G. Kadianakis, M. Garnett, M. Taiwo, and A. Dietrichs, "EIP-4844: Shard Blob Transactions," https://eips.ethereum.org/EIPS/eip-4844.
[17] J. Neu, "Data Availability Sampling: From Basics to Open Problems," https://www.paradigm.xyz/2022/08/das, Aug. 2022.
[18] J. Groth, "On the size of pairing-based non-interactive arguments," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2016, pp. 305–326.
[19] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptology ePrint Archive*, 2018.
[20] F. G. Figueira, M. Derka, C. L. Chiu, and J. Gorzny, "A practical rollup escape hatch design," *arXiv*, 2025.
[21] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM TOCS*, 2002.
[22] U. W. Chohan, "Proof-of-stake algorithmic methods: a comparative summary," *SSRN 3131897*, 2018.
[23] E. Buchman, "Tendermint: Byzantine Fault Tolerance in the Age of Blockchains," *PhD Thesis*, 2016.
[24] S. Cho, R. Fontugne, K. Cho, A. Dainotti, and P. Gill, "BGP hijacking classification," in *Proc. IEEE TMA*, 2019.