Continuum: A Cryptographic Layer for FIFO Ordering at microsecond resolution

0xSeldon *seldon@fermilabs.xyz*

June 15, 2025

Abstract

Blockchain systems today face fundamental challenges in transaction sequencing and latency. Traditional block-based protocols suffer from temporary centralization where block proposers have unilateral control over transaction ordering, enabling miner extractable value (MEV) extraction and frontrunning attacks. We present Continuum, a novel transaction sequencing protocol that eliminates blocks in favor of a continuous-time ledger with cryptographically enforced FIFO ordering. Continuum uses a verifiable delay function (VDF) as a decentralized clock, producing timestamps every 100 microseconds and anchoring each transaction to a specific time. This design removes the sequencer's ability to arbitrarily reorder transactions, dramatically reducing MEV. Combined with VDF linked timelock encryption, content based reordering, exclusion, or frontrunning becomes structurally impossible. We provide formal security analysis, demonstrate practical performance parameters, and discuss integration pathways for rollups, DeFi applications, and cross-chain systems. We also outline how arbitrary execution logic (e.g. EVM-compatible state machines, DEX order matching, rollup block processing) can be layered atop Continuum's ordered transaction log, enabling trustless consumption of the ordering service. The result is a blueprint for a high-speed, fair transaction ordering layer that mitigates Miner Extractable Value (MEV) and can serve as a foundation for scalable and provably fair decentralized applications.

1 Introduction

First-in First-Out (FIFO) is the gold standard ordering rule used by traditional centralised CLOBs, that handle the bulk of global transaction volumes. FIFO at a very short time resolution (< 1ms) is an essential component for a CLOB to serve as the venue for true price discovery; if a coarser ordering rule is used, real-time price discovery is impossible on such a platform. In today's DEXes, no implementation of cryptographic FIFO has ever been made - exchanges either rely on purely trusted FIFO (eg., centralised, "black-box" rollup sequencers), or use work-arounds for coarser time resolution (eg. cancels-before-takes rules to compensate for 70ms block time on Hyperliquid, or 400ms block time on solana). We posit that true, verifiable, sub millisecond resolution FIFO is the integral missing step to bringing price discovery on chain.

In a recent essay, MEV was identified as the key limitation on scalability of blockchains. This is because the prevalence of MEV-linked spam transactions rises linearly/super-linearly with throughput [2]. In traditional block-based protocols, each block proposer (miner or validator) has unilateral control over the ordering of transactions within their block. For example, in Ethereum, block producers can reorder or insert transactions to capture arbitrage profits (a form of frontrunning) [6]. This leads to priority gas auctions and other games that undermine fairness and even threaten consensus security. MEV arises because of a dirty secret: most blockchains are temporarily centralized, and a single leader decides transaction order in each block. Solutions like Flashbots have introduced off-chain MEV auctions to make extraction more transparent, but they do not fundamentally prevent unfair ordering or frontrunning (they mainly aim to "democratize" MEV, not eliminate it) [9].

Even newer high-throughput architectures still rely on batched blocks and trust in leaders. Solana's Proof of History (PoH), for instance, uses a continuous hash chain as a cryptographic timestamp to order events, achieving sub-second confirmation times and high throughput. However, Solana still produces discrete blocks and relies on a Proof-of-Stake consensus to finalize them; ordering within blocks remains discretionary [18]. Layer-2 rollups like Optimism and Arbitrum introduce a dedicated sequencer that orders transactions off-chain and posts them to layer-1, but this merely shifts extractable value to the rollup operator (sometimes called Rollup Extractable Value, REV) and introduces new latency [3]. In optimistic rollups, users often wait up to one week for trustless finality of transactions, and even to observe the latest state the community must wait until the sequencer's batch is posted on L1. Chainlink's Fair Sequencing Service (FSS) proposes to improve fairness by having a decentralized oracle network reach consensus on transaction order (removing the single leader). FSS (and related academic works like Aequitas [12]) enforce order-fairness via multi-party agreement, but at the cost of added complexity and consensus overhead on the order of several seconds [5, 10].

Continuum takes a different approach: it provides a minimal transaction sequencing layer that operates continuously in time, enforced by cryptography rather than by rotating block leaders or multi-party voting. In Continuum, there are no blocks – each transaction is individually committed to a globally ordered log as soon as it arrives. A single designated sequencer (which can rotate or be replaced via committee consensus or governance outside the scope of this core protocol) receives transactions from users and immediately timestamps and commits each one to the log. The timestamping mechanism is a verifiable delay function (VDF) acting as a decentralized clock: the sequencer must compute a new VDF output every fixed interval (e.g. every $\Delta = 100$ us), and each transaction's commit includes the latest VDF output as a time anchor [1, 16]. By linking each transaction's hash to the previous transaction and to the current VDF output, Continuum forms an immutable hash/Merkle chain of transactions with cryptographically verifiable timestamps at millisecond granularity.

This design yields several key benefits:

- **Trustless Time and Order:** The VDF acts as a global, trustless clock that proves a minimum delay between events. A VDF is a function that takes a prescribed sequential time to evaluate and yields an output that can be efficiently verified [1]. Even with massive parallel computing, it cannot be computed faster than real time (Δ per tick). Thus, the sequencer cannot compress or rearrange time: if transaction A is processed before transaction B in real time, A will inevitably receive an earlier VDF timestamp than B, enforcing a form of chronological order integrity. In effect, time itself (via the VDF) serves as the arbiter of order, rather than the whims of a block producer.

- **Continuous Ledger (No Batching):** By committing transactions one-by-one in very fine-grained ticks, Continuum avoids batching many transactions into a single block. This significantly reduces latency – users no longer wait for the next block proposal slot; a transaction can be confirmed in the next 100 us tick. It effectively creates an append-only log with a verifiable passage of time between events, similar in spirit to Solana's PoH ledger but with a cryptographically verifiable delay function rather than an ad-hoc hash chain.

- **MEV Mitigation:** Because transaction ordering is constrained by an external time beacon, the sequencer's ability to arbitrarily reorder, insert, or front-run transactions is dramatically curtailed. The sequencer no longer has free rein to permute transactions within a block to its advantage; any attempt to shuffle transactions out of timestamp order would violate Continuum's validity rules and be publicly detectable. This approach tackles the root cause of MEV by removing the "time-free" slack that block proposers normally exploit to reorder transactions. In short, Continuum removes the temporary centralization of block building – the sequencer is tightly bound by the cryptographic clock to maintain the faithful arrival order of transactions.

- **Fast Finality:** Continuum provides fast and quantifiable finality. It ties settlement to an explicit unit of time-work: a 100 us VDF tick. To reverse or re-order a transaction that has been buried under n subsequent ticks, an attacker would have to (a) recompute $n+1$ sequential VDF steps faster than the honest network can (i.e. catch up after falling behind), and (b) propagate a conflicting chain worldwide before the honest chain advances further. With today's high-performance VDF hardware (ASICs for class groups [14]), this amounts to roughly one millisecond of irreversibility per tick. In practice, co-located participants can treat a depth of $\sim$10 ticks ($\approx$10 ms) as nearly certain finality (faster than a human blink), and a depth of 50–150 ticks ($<$0.2 s) provides settlement finality that outpaces the speed-of-light round-trip between any two global data centers. For context, Ethereum's Proof-of-Stake requires two epochs ($\sim$12 minutes) to reach slashing-enforced finality [7], optimistic rollups use $\sim$7-day dispute windows, and even Solana's recent proposals aim for $\sim$150 ms world-wide confirmation. Continuum narrows global finality to the sub-second range and local confidence to single-digit milliseconds—an order-of-magnitude improvement over the fastest existing L1s—while retaining transparent, cryptographically auditable settlement.

In summary, Continuum is a blueprint for a high-speed, fair transaction ordering layer that mitigates MEV and can serve as a foundation for scalable and provably fair decentralized applications. The remainder of this paper details the Continuum protocol design, analyzes its security and performance, and discusses how it can integrate into various blockchain ecosystems.

2 Continuum Protocol Design

2.1 VDF Time Beacon and Millisecond Ticks

Continuum's core innovation is the use of a continuous verifiable delay function as a time beacon to impose a global ordering cadence. Specifically, an instance of a Wesolowski VDF [16] is run continuously by the sequencer, producing a new output every fixed time interval Δ (target 100 us). Each VDF output is essentially a proof-of-time: it proves that a certain minimum wall-clock interval has elapsed since the previous output. We denote by x_t the VDF output at tick t, and π_t the accompanying proof. The VDF has the property that given the previous output x_{t-1}, one must perform a predetermined number n of sequential operations (squarings in an RSA group, in Wesolowski's construction [16]) to obtain x_t; no parallel or shortcut method can produce x_t faster than real time. This sequential-work requirement means that the sequencer cannot produce outputs faster than one per Δ in real time – the protocol enforces a cryptographic heartbeat of 100 us per tick.

In practice, the sequencer initializes the VDF with some seed x_0 (for example, the output of a randomness beacon or a genesis value). Then for each tick $t = 1, 2, 3, \ldots$, the sequencer computes:

- $x_t = \mathrm{VDF}(x_{t-1})$, which takes $\sim$100 us by design, and
- π_t, a short proof that x_t was computed correctly from x_{t-1}.

Everyone can efficiently verify π_t to confirm that the sequencer indeed spent the required time Δ between ticks. To improve efficiency, Continuum can use an aggregated VDF proof technique [8], wherein one combined proof attests to many sequential outputs (e.g. a proof per 1000 ticks) instead of publishing a proof for every single tick. Prior research on continuous VDFs [8] and batched verification of VDFs [15] provides methods to compress the proof overhead. In essence, the VDF outputs serve as an unstoppable clock: as long as at least one honest party (the sequencer) is computing the VDF, time ticks forward and can be verified by all.

2.2 Transaction Commitments and Hash-Linked Chain

Each transaction T_i in Continuum is immediately anchored to the current VDF tick. When the sequencer is ready to commit a new user transaction (say a client transaction arrives or is next in the queue), the sequencer waits until the next VDF tick t is reached (ensuring the fixed time cadence) and then creates a cryptographic commitment that links the transaction, the time, and the prior state of the ledger. Formally, let $H(\cdot)$ be a cryptographic hash function. We define C_i as the ledger state commitment after i transactions – essentially a running state root or fingerprint of the transaction log up to and including T_i. We initialize C_0 to a known genesis value (for example, the hash of the initial VDF seed x_0 concatenated with an empty ledger state).

Now for each new transaction T_{i+1} that the sequencer processes at tick t, the sequencer obtains the latest VDF output x_t (and its proof π_t). The sequencer assigns timestamp t to T_{i+1} and forms the new state commitment by hashing together the previous commitment, the new transaction, and the time beacon output:

$$C_{i+1} := H\left(C_i \parallel T_{i+1} \parallel x_t\right). \tag{1}$$

This simple construction creates a hash-linked chain of transactions: each C_{i+1} cryptographically links to the prior state C_i, so any attempt to remove or reorder transactions will break the chain. Additionally, each commitment incorporates the VDF output x_t, which is rooted in time. We can interpret x_t as a time root and C_{i+1} as a sort of Merkle-tree node that combines the prior state and the new transaction under that time root. In essence, each transaction is linked both to the latest VDF tick and to the previous transaction's state, yielding a time-anchored transaction log.

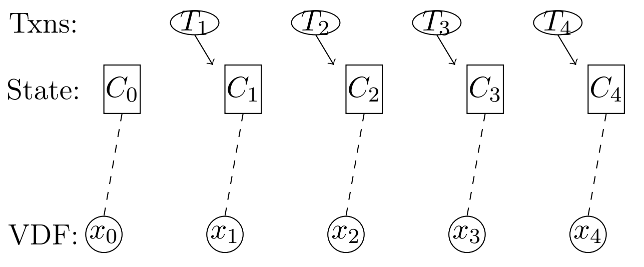

Figure 1: Illustration of Continuum's continuous VDF chain, incorporating transaction commitments at every 100 us tick. Each new transaction (top row, labeled "Txns") is processed at the next available tick and is immediately committed by hashing it with the previous state root and the current VDF output (bottom row, $x_0, x_1, \ldots$). Each commitment C_{i+1} thus links to the entire history before it (C_i) and to the time x_t, enforcing that transactions follow the strict time order of the VDF ticks.

In cases where no user transaction is available for a given tick, the sequencer must still progress the VDF and publish the tick. The sequencer can either include a no-op (empty) transaction at that tick or simply advance the state commitment by hashing the prior commitment with the new x_t (essentially a dummy state update). What it cannot do is skip ticks or stretch the time interval. There is a heartbeat every Δ milliseconds, whether or not a real transaction is included. This design ensures a prover cannot halt or slow down time to wait for some advantage (e.g. pausing to see another market event before deciding what transaction to include). In effect, Continuum provides a trustless timestamping service: every transaction comes with a provable timestamp (the tick number) and an implicit ordering guarantee that no later transaction can be placed before an earlier timestamp.

2.3 Separation of Sequencing and Execution

Continuum focuses solely on ordering transactions in a fair, verifiable timeline. Importantly, it does not prescribe any particular transaction format or execution logic. In other words, Continuum cleanly separates the sequencing layer (ordering consensus) from the execution layer (state machine updates). The Continuum sequencer simply takes transactions (opaque blobs from its perspective), assigns timestamps and positions to them, and outputs an ordered log with state commitments. What those transactions do or how state is updated as a result is left to a higher layer.

This means one can layer arbitrary deterministic execution logic atop Continuum's ordered log. For example, the transactions could be Ethereum-like (with smart contract calls and an EVM state), and an Ethereum client could process the Continuum-ordered list of transactions to update account balances and storage. Or the transactions could be orders in a decentralized exchange (DEX) system, and a matching engine could execute trades in that timestamp order. Because Continuum provides a tamper-proof sequence and timing for each input, any execution layer can consume this sequence and know that the order is globally verifiable and fair. This plug-and-play design makes Continuum a general sequencing module that consensus protocols, rollups, and DApps could use for ordering, while they remain free to define their own state transition functions.

Concretely, a Continuum full node (verifier) would do two things: (1) verify the ordering log (check VDF proofs, hash chain integrity, and timestamp monotonicity as described below), and (2) feed the transactions into the application's state transition function in that order to compute the resulting state. The crucial point is that step (2) is application-specific and outside Continuum's minimal responsibilities. This approach is analogous to proposals that separate consensus from execution (such as Ethereum's proposer-builder separation or various off-chain execution environments): Continuum handles consensus on when and in what order things happen, but not what those things do. Such modularity also opens the door to using zkSNARKs or rollup techniques on the execution side, if desired, to prove the correctness of state updates, while Continuum itself ensures the ordering is fair and fixed.

2.4 Formal State Transition Rules

We now summarize Continuum's state transition rules and validity conditions more formally. The system can be viewed as a state machine that appends one transaction at a time, with an enforced time gap between each append. Each state transition (each appended transaction) must satisfy certain validity conditions for the chain to be considered valid. These conditions ensure the cryptographic linkage and time-ordering properties are upheld:

1. **Monotonic Timestamps:** If transaction T_i has timestamp t_i and T_j has timestamp t_j in the log, then $t_i < t_j$ implies T_i appears before T_j in the log. In other words, timestamps strictly increase along the chain, and no transaction can appear with a timestamp earlier than a previous one. (If multiple transactions are submitted within the same 100 us interval, the sequencer can include at most one of them in that tick; others will be assigned subsequent ticks, preserving a total order.)

2. **Hash-Chain Integrity:** Each state commitment is correctly derived from the previous state, the transaction, and the tick's VDF output. Formally, for each consecutive pair of transactions T_i and T_{i+1} with respective ticks t_i and t_{i+1}, the commitment must satisfy $C_{i+1} = H(C_i \parallel T_{i+1} \parallel x_{t_{i+1}})$ as defined. This links the chain of states and ensures no transaction or time can be removed or altered without breaking the hash.

3. **Correct VDF Sequence:** For each transaction T_j with timestamp t_j, the VDF output x_{t_j} included in its commitment must be valid relative to the previous transaction's tick.

That is, $\text{Verify}(x_{t_{j-1}} \oplus H(T_j), 1, x_{t_j}, \pi_{t_j}) = \text{true}$, meaning that running one tick of the VDF (or the configured number of sequential steps n) from the prior output (optionally mixed with the new transaction's hash) yields x_{t_j}. Moreover, if there were any idle ticks with no transactions between $t_{j-1} + 1$ and t_j, the sequencer must still have computed and published all intermediate VDF outputs for those ticks (preventing it from "skipping" ahead). In short, the VDF chain must advance one step per tick, continuously, even if some ticks carry no transactions.

4. **Timely Timestamp (No Postdating):** The real-world time when a transaction T_i was received by the sequencer must be $\leq$ the real time corresponding to its tick t_i. In other words, a sequencer cannot assign a future timestamp to a transaction that arrived now (no postdating to make a transaction seem earlier than it was). This ensures honesty in reporting when the transaction was processed. Conversely, if a transaction waits in a queue, it will simply get a later tick (no backdating is possible). All honest nodes can check that timestamps reflect a valid progression of time relative to when they themselves saw transactions arrive (this condition is more subtle and typically relies on assuming the sequencer's clock is roughly synchronized with wall-clock time or network propagation delays are bounded).

If any of these conditions is violated – e.g. a hash doesn't match, or a VDF proof is wrong, or a transaction's timestamp goes backwards or is out of range – then the transaction log is invalid and can be rejected by any verifying node. These rules guarantee that any accepted Continuum chain has a strictly increasing time base and an unbreakable chain of commitments, making it cryptographically impossible for the sequencer to reorder transactions once they are committed.

3 Security and Liveness Analysis

We now discuss Continuum's security properties and liveness guarantees. In a setting with a single designated sequencer, some of these properties (like liveness) hold under the assumption that the sequencer is honest and online. We later describe how to extend the protocol to a decentralized sequencer committee to handle failures or malicious behavior. Here we focus on the guarantees provided by the core continuous-time protocol, assuming at least one honest sequencer drives the clock.

3.1 Order Integrity

Order Integrity means that if one transaction is supposed to happen before another in time, it indeed appears before in the ledger. Continuum achieves strict chronological ordering: if transaction T_i has an earlier timestamp than T_j (i.e. $t_i < t_j$), then no valid chain can place T_j ahead of T_i. Equivalently, it is impossible for a valid Continuum log to invert the order of two transactions with different timestamps.

[Order Integrity] In Continuum, if two transactions T_i and T_j have timestamps t_i and t_j with $t_i < t_j$, then T_i will appear before T_j in any valid transaction log. It is infeasible for the sequencer to produce a valid log where this order is violated.

[Proof Sketch] By design, timestamps are strictly increasing along the chain (Validity Rule 1 above). The sequencer assigns timestamps in increasing order as real time advances. Suppose for contradiction the sequencer published a chain where T_j (timestamp t_j) appears before T_i (timestamp t_i) even though $t_i < t_j$. Then at the point in the log where T_j appears, the chain would show a later tick followed by an earlier tick – violating the monotonic timestamp rule. The VDF proof would also fail: T_j's commitment would include x_{t_j} which corresponds to a time after T_i's x_{t_i}, yet T_i is placed later. This inconsistency would be caught by any verifier checking the sequence of VDF outputs and timestamps. Therefore such a log cannot be valid. Essentially, the cryptographic clock enforces that time moves forward in the ledger, and the ledger order must respect real-time order.

In Continuum, order integrity holds automatically as long as the VDF assumption holds (no one can speed up the clock). A dishonest sequencer cannot get around this by manipulating network propagation or withholding earlier transactions: even if the sequencer tries to censor a transaction that arrived earlier and include a later one first, the earlier transaction will have a lower timestamp (since it arrived to the sequencer's mempool first) and any attempt to include it afterward with a lower timestamp would immediately reveal the misconduct (the timestamp order would violate the chain). Thus, order-fairness is built-in—Continuum provides receive-order fairness by tying ordering to time of receipt.

3.2 Non-Equivocation

Non-equivocation means the sequencer cannot present two different transaction logs (two histories) to different observers without being detected. In a decentralized context, this is crucial: we want to ensure there is effectively one single canonical history at any given time, and a malicious sequencer cannot fork the ledger or double-spend by giving conflicting sequences to different users.

Continuum's design inherently limits equivocation because every state commitment is hash-linked and time-linked. If the sequencer tries to produce two distinct logs that diverge at some transaction, one of those logs will contain a different sequence or different time stamps, which will either fail verification or at least yield two different state commitments for the same prior state.

[Non-Equivocation] There is at most one valid Continuum chain (transaction log) extending from a given genesis state at any given time. In particular, the sequencer cannot produce two distinct valid state transitions or logs that share the same previous state C_i but lead to different C_{i+1} values.

[Rationale] If the sequencer attempted to fork, consider the point of divergence: up to some C_k the logs are identical, and then the sequencer produces two different next transactions T', T'' (or perhaps the same transaction but in different orders later). Because the commitment includes the entire prior hash C_k and the VDF output x_t, any two different continuations will result in two different hashes $C'_{k+1} \neq C''_{k+1}$. An honest full node or light client seeing both would detect that the sequencer equivocated. Moreover, since the sequencer is supposed to sign the commitments or provide VDF proofs, any conflicting log segment (two different sequences with a common prefix) can serve as undeniable evidence of equivocation. In practice, one can enforce non-equivocation by slashing: if anyone presents two valid Continuum logs that share a prefix but diverge at some point, the sequencer (or its key) is proven to have misbehaved and can be penalized by the protocol. Under the assumption that cryptographic hashes cannot be forged and the sequencer cannot outrun the VDF, it cannot maintain two divergent chains undetected. Thus, all honest observers will converge on a single chain.

3.3 Liveness and Throughput

Liveness means that the system continues to process transactions and the chain keeps growing, assuming honest participants. In the Continuum design with a single sequencer, liveness is straightforward if the sequencer remains online and active: the sequencer will keep computing the VDF and including any incoming transactions at the next ticks. The throughput is essentially fixed at $1/\Delta$ transactions per second in steady state. With $\Delta = 1$ ms, this corresponds to a maximum throughput of 1000 transactions per second if every tick carries a transaction. Continuum is actually designed to target an even higher throughput: by allowing up to one transaction per tick, it can reach $\rho^* = 10^4$ tx/s (10,000 TPS) if the hardware and network can support every tick being filled. This is several orders of magnitude above Ethereum's ∼15–30

TPS and even above typical rollups (which in practice handle a few hundred TPS at most). It is comparable to or higher than Solana's typical throughput ($\sim$1,000–3,000 TPS under real-world conditions, even though Solana's theoretical limit is higher).

Under normal operation (honest sequencer), Continuum's liveness is limited only by the hardware's ability to keep up with the 100 us VDF cycle. Current VDF ASICs indicate that $\sim$100 us per tick is achievable with room to scale as hardware improves [14]. The network and the sequencer's transaction pipeline must also handle 10k TPS throughput (which implies efficient mempool and networking, but this is comparable to existing high-performance chains). If the transaction load is lower, the sequencer simply produces empty ticks as needed (the chain still progresses in time, preserving liveness of the clock). Finality, as discussed, is practically achieved within milliseconds to sub-second depths, so users get very fast confirmation that their transaction is irreversibly included.

The main threat to liveness in a single-sequencer model is if the sequencer goes offline or maliciously stops processing transactions (a censorship or halt attack). In such a case, the chain's progress would stall since no new ticks are being produced. Continuum's core protocol does not handle leader failure by itself (it assumes an honest available sequencer), but in a broader deployment one would introduce a mechanism to rotate or replace the sequencer when needed.

3.4 Finality

Continuum provides probabilistic finality with an exceedingly steep security curve due to the time-based nature of the chain. Once a transaction has been included and a bit of time passes, the probability that it could be removed or rolled back becomes essentially zero (barring a massive break in cryptographic assumptions or the emergence of super-fast hardware beyond our assumptions). In this sense, Continuum's finality is akin to the physical finality of proof-of-work chains but on a dramatically accelerated timeline.

To quantify finality, consider that an attacker who wants to revert a transaction with k confirmations (i.e. k ticks followed it) would need to produce an alternative chain that is $k+1$ ticks longer (to convince others to adopt it) and do so in less time than the honest chain took to produce k ticks. Under the VDF sequentiality assumption, this is infeasible unless the attacker has a significantly faster sequential computing ability than the honest network. With a 100 us tick, even being one tick behind means the attacker is 100 us of sequential work behind. There is no notion of probabilistic block races as in PoW; here you either can outrun time or you cannot. If you cannot, then once a transaction is one tick deep, it cannot be beaten by a competing sequence unless cryptography is broken.

We could define a notion of ϵ-finality: after k ticks, the chance that a conflicting log could appear (without detection or slashing) is $\leq \epsilon$. Under the assumption that no faster VDF computation exists than what honest nodes have, for any $k \geq 1$, we effectively have $\epsilon \approx 0$ (since any conflict would require a violation of the VDF security). Even under slightly weaker assumptions (perhaps the attacker has a marginally faster machine, say 5% faster), the chain's finality still becomes exponential in k because the attacker falls further behind each tick. In practice, finality in Continuum is achieved after on the order of tens of milliseconds globally – an unheard-of level in prior consensus systems.

It's worth noting that finality in Continuum assumes the sequencer is eventually honest (or that a malicious sequencer is detected and stopped). If the sequencer itself is malicious, it could censor or withhold transactions (affecting liveness, as discussed) but it still cannot revert history that has been published without immediate evidence. The worst a malicious sequencer can do to finality is halt the chain (stop producing ticks) or try to equivocate (publish two different chains), which would be caught and lead to punishment. But once a transaction is posted and some time passes (even a second), not even the sequencer can go back and remove it, because any alternative history would require turning back the VDF clock, which is computationally infeasible. Thus, Continuum provides a very strong notion of finality: even the single leader cannot undo its past commitments after the fact (assuming a vigilant community that checks for equivocation attempts).

4 Integration and Use Cases

Continuum is a general sequencing layer that can be utilized in many blockchain contexts. Here we outline how it can integrate with various systems and applications to provide fair ordering and timestamping.

4.1 Layer-2 Rollups and Sequencers

Many layer-2 rollups today (Optimism, Arbitrum, etc.) use a centralized sequencer to order user transactions off-chain before compressing them and posting to Ethereum. This introduces centralization and fairness issues: the rollup operator can extract MEV (e.g. via reordering user trades, a phenomenon sometimes dubbed Rollup Extractable Value), and users must trust the sequencer's ordering until the batch is on L1 [3]. Continuum can act as a drop-in replacement for a rollup's sequencer, turning the sequencing process into a decentralized, time-audited service.

In this model, the rollup operators (or validators) would collectively run a Continuum sequencer. User transactions destined for the rollup are sent to the Continuum network, which orders them fairly in real time. The ordered list (with timestamps) is then used to build the rollup block. The rollup's state transition (whether via fraud proofs in an optimistic rollup or validity proofs in a ZK-rollup) is deterministic given the transaction order, so the integrity of the rollup is maintained. The key difference is that now the transaction order is verifiable and unbiased: the rollup sequencer can no longer arbitrarily insert or reorder transactions because any deviation from arrival-order would break the Continuum log's validity. If the rollup sequencer attempts to produce two different L2 histories (equivocate) or include out-of-order transactions, it would be caught (and could be slashed if Continuum is run in a PoS setting).

Continuum can also label transactions by rollup chain ID (if multiple rollups share one Continuum instance), allowing a single global sequencer to serve many chains. The rollup operators simply take the subset of transactions for their chain from the global log and process them. This shared sequencing layer approach has been discussed in Ethereum research – using a base layer as a global ordering service for many rollups to enforce fairness. Continuum provides exactly that: a decentralized global clock and ordering service that all rollups could tap into. It even offers a built-in cryptographic audit trail.

Another benefit for rollups is fast finality on L2. With Continuum, users see their transactions ordered and finalized in milliseconds on the sequencing layer, rather than waiting for an L1 batch inclusion. This could significantly improve user experience, while L1 is used mainly as a security anchor for proofs.

4.2 Decentralized Exchanges (DEXs) and DeFi

One of the primary motivations for fair sequencing is to prevent unfair trade ordering on DEXs and DeFi applications (frontrunning, sandwich attacks, etc.). In a traditional AMM or order-book DEX on Ethereum, miners or arbitrage bots can exploit the ordering of transactions to their advantage, leading to MEV extraction that harms regular users [6]. By integrating with Continuum, a DEX can ensure that all user orders are processed in the exact order they were received (or committed) in time, eliminating many of these attacks.

For example, consider an automated market maker (AMM) like Uniswap. Instead of users sending swap transactions to Ethereum mempool (where they might get reordered by miners based on gas fees), users would submit their swaps to the Continuum sequencer network. Continuum would timestamp and order these swap transactions fairly by arrival time. The DEX's smart contracts on Ethereum could be modified such that they accept an ordered batch of swaps (perhaps via a rollup or batch submitter) that are already sorted by Continuum timestamps. The contract could even verify the Continuum proofs or require a Continuum reference for each trade. The trades would then execute in that fair order, preventing any user from jumping ahead by paying a higher fee (since Continuum does not allow fee-based reordering). Essentially, it creates a fair ordering service as an oracle to the DEX. Empirical analyses of DEX MEV (e.g. Flash Boys 2.0 [6]) highlight how damaging the current ordering games are; Continuum offers a way to neutralize that by making the order deterministic and verifiable.

Another use-case is atomic arbitrage prevention: If multiple exchanges (or pools) use Continuum, then a bot cannot slip in an arbitrage transaction between two user transactions within the same millisecond, because everything is globally ordered at 100 us granularity. Continuum's commit-reveal scheme (discussed later) can further hide transaction details to prevent even more subtle timing exploits.

In DeFi more broadly, any protocol where the ordering of user actions matters (auctions, liquidations, oracle updates) could benefit from a Continuum ordering feed. Continuum can serve as a decentralized clock network for DeFi, giving every transaction a global timestamp that contracts can reference to enforce ordering rules. For instance, a lending platform could specify that liquidations must be processed in order of when the shortfall occurred, using Continuum timestamps to sort liquidation calls. This would remove the incentive to spam the network with high fees to win liquidation races – whoever's transaction was first in time (as per Continuum) gets to liquidate, period.

4.3 Cross-Chain Bridges and Settlement Ordering

Cross-chain bridges often face MEV-like issues as well. For example, if users are moving funds between chains, the order in which bridge transactions are executed can be exploited (imagine a scenario where an attacker can reorder withdrawals to profit from price changes or arbitrage differences across chains). A malicious bridge operator could delay or reorder cross-chain transactions for profit. By using Continuum as a global ordering base layer, bridge actions across multiple chains can be anchored to one fair timeline.

For instance, suppose multiple blockchain networks agree to use Continuum for ordering bridge events. All bridge transactions (lock, release, swap across chains) could be sent to Continuum. Continuum would produce a single ordered log of these events. Each chain's bridge contract would then execute the events in the Continuum order. This ensures consistency and fairness across chains – no one can exploit the sequencing to gain an advantage on one chain versus another, since the ordering is fixed globally. It effectively provides a cross-domain MEV resistance: an attacker cannot reorder events on one chain differently from another to, say, exploit a price difference, because the order is locked in Continuum's ledger.

Similarly, cross-chain settlement systems (like coordinated batch auctions that settle across several DEXs on different chains) could use Continuum to agree on a single sequence of trades that apply to all. This would prevent participants from, for example, executing a trade on one exchange slightly before another in an inconsistent order. Continuum would guarantee a unified event ordering, simplifying the logic needed to keep multiple systems in sync.

4.4 On-Chain Timestamped Contracts

Continuum's verifiable timestamps can also be fed into L1 smart contracts as a source of trustless time. For example, Ethereum smart contracts currently have to rely on block timestamps (which are coarse and manipulable within a certain bound by miners) or external oracles (which add trust assumptions). With Continuum, one could imagine a contract that requires actions to be accompanied by a Continuum timestamp proof. If the timestamps are not strictly increasing or if an action is out-of-order, the contract could reject it.

One concrete pattern: a contract on Ethereum could accept a batch of transactions (or commitments of transactions) with Continuum proofs that they are in sorted order by time. The contract would then apply them. If someone attempted to feed two transactions out of order, the proofs would not check out and it could trigger a penalty. This way, even on a base chain that doesn't natively implement Continuum, one can enforce fair ordering at the application level by leveraging Continuum off-chain and verifying it on-chain. Chainlink's FSS approach similarly considered using oracle networks to provide ordering; Continuum provides a more cryptographically direct approach (no committee voting, just proofs of time).

Overall, Continuum aims to serve as a public utility for time and order in blockchain systems, analogous to how a global clock provides a common reference for events. Any system that needs a reliable ordering of events or a notion of timestamp can benefit from anchoring to Continuum's timeline.

5 Advanced Features

We now describe several advanced features and extensions of the core Continuum protocol that enhance its fairness, privacy, decentralization, and adaptability.

5.1 Commit-Reveal Scheme for Fairness and Privacy

While Continuum as defined above removes the ability to reorder transactions between ticks (inter-tick permutation), a savvy sequencer could still exploit transaction contents within the same tick. For example, if two transactions arrive almost simultaneously, the sequencer might see that one is very lucrative (e.g. a large trade) and decide to include a small "back-run" transaction right after it in the next tick, or choose how to act on it in the subsequent ticks. Also, a malicious sequencer could censor a transaction it knows about, or adjust its own orders knowing what users are doing. To mitigate these residual vectors of MEV, Continuum can incorporate an opt-in commit-reveal scheme that hides transaction payloads until a future time, ensuring the sequencer orders transactions without being able to determine their content (thus preventing content-based manipulation like selective frontrunning or censorship based on transaction details).

The scheme works as follows. Suppose Alice has a transaction T that she wants to submit without revealing it immediately. She chooses a future tick $N + \delta$ (δ is a delay, e.g. $\delta = 5$ ticks or 5 ms) at which she wants her transaction to be readable/executable. She then encrypts T under a key that is time-locked by the VDF. One way to do this is: Alice generates a random symmetric key K, and encrypts her transaction with K (e.g. using AES-GCM for authenticated encryption) to obtain ciphertext C. She then derives K in a way that is tied to the VDF chain: for instance, she could publish a value that will yield K after δ VDF iterations. Concretely, Alice can take a random secret s, and compute $y = \mathrm{VDF}^{\delta}(s)$ by running δ squarings of the VDF function on s. The value y is not computed fully by Alice (since δ steps may be expensive for her), but she knows that after δ ticks, y can be derived by anyone. She then derives $K = H(y)$ (a hash of y), and uses K to encrypt T. She submits (C, some proof of s or commitment to s, and δ) to the sequencer.

When the sequencer receives this, it treats it as a committed transaction. It cannot decrypt C because deriving K requires δ sequential steps of the VDF, which by design will take δ ticks of time. The sequencer therefore just includes the ciphertext C in the log at the next tick (assigning it a timestamp). This gives Alice's transaction a position in the order (relative to other transactions) without revealing what the transaction actually is. After the inclusion, the clock keeps ticking. Once tick $N + \delta$ is reached, the VDF output for that time (or Alice herself, or any observer) will effectively produce the key K (since after δ ticks, $y = \mathrm{VDF}^{\delta}(s)$ is now computable). At that point, C can be decrypted to reveal the plaintext transaction T. The network (or the sequencer) can then process T and incorporate its effect into the state.

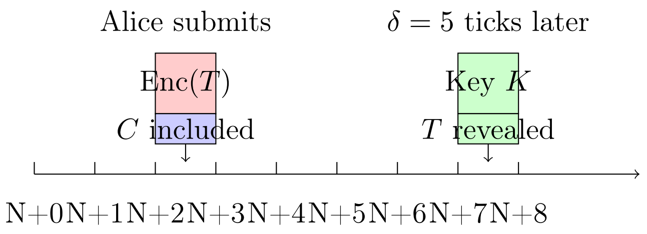

Figure 2: Time-lock commit-reveal scheme in Continuum. In this example, Alice submits an encrypted transaction at tick N+2, designed to remain secret until tick N+7. The sequencer, unaware of the transaction's contents, includes the encrypted transaction in the log at tick N+2 (assigning it a timestamp N+2). After enough time passes (tick N+7), the encryption key is automatically derived via the VDF tick chain, and the transaction is decrypted and executed in the sequence.

This commit-reveal approach ensures ordering fairness: the sequencer must commit to an ordering of transactions without knowing certain transactions' details. It can't favor or disfavor a transaction based on its content if it doesn't know what that content is. The cryptographic time-lock (provided by the VDF) guarantees that the secret will be revealed only after the transaction is secured in place (and enough ticks have passed to finalize its position). Even if the sequencer tried to delay the reveal, it can't – the reveal is tied to the deterministic progress of the VDF. All cryptographic secrecy here is derived solely from the sequential hardness of the VDF (no additional trusted randomness beacon is required).

In practice, not every transaction needs this treatment (since it does introduce a slight delay for execution). High-value or MEV-sensitive transactions could opt-in to commit-reveal, whereas regular transactions can be plain. The protocol would treat encrypted payloads as placeholder transactions that later get filled in. One must ensure that once the plaintext is revealed, it indeed matches what was committed (integrity check via the encryption nonce or a hash commitment). Techniques like symmetric key encryption with a publicly verifiable commitment (so the sequencer can't cheat by including gibberish) are used. The scheme described is similar to proposals in FSS and other fair ordering literature, but Continuum's unique advantage is that the time-lock is natively provided by the continuous VDF tick chain. In other words, the same mechanism that provides time and order also enables privacy until a preset time.

5.2 Decentralized Sequencer Rotation (VDF+PoS)

So far, we have considered a single sequencer for simplicity. In a production setting, we want Continuum to be permissionless or at least decentralized – meaning multiple participants can compete or take turns to run the sequencer, rather than a fixed trusted node. We outline here a leader election and committee mechanism based on Proof-of-Stake and VDFs that allows the sequencer role to rotate each epoch (a period consisting of many ticks, say on the order of seconds or minutes). The goals are to maintain the 100 us tick rate, ensure continuity of the VDF chain across leader transitions, and have built-in incentives and slashing for performance and honesty.

Stake and Eligibility: Participants (validators) stake a certain amount of tokens to be eligible as sequencers. To join, a validator must prove it has the hardware capable of keeping up with the 100 us VDF. For example, a joining validator could be required to produce an attestation VDF proof: it takes the current VDF output and runs, say, n squarings (the work of one tick) many times in a row (perhaps simulating an epoch's worth of ticks), then provides a proof of this computation. If the time it took is within, say, 5% of the network's reference time for that many ticks, the validator is deemed fast enough. This ensures that every active validator has proven it possesses hardware that can sustain the required tick rate. In essence, the entry criteria enforce that no validator with sub-par hardware slows down the system.

Leader Lottery: At the end of each epoch, a new leader (sequencer) is chosen randomly from the validator set, weighted by stake. Continuum can achieve this leader selection by using the VDF outputs and collective signatures as a randomness beacon. For instance, suppose an epoch lasts T_e ticks. At tick T_e, the current leader publishes a final digest of the epoch (e.g. $D_e = H(C_{\text{last}} \parallel x_{T_e})$, the hash of the last state commitment and last VDF output). All validators then collectively sign D_e using BLS signatures. The signatures are aggregated into σ_e, and the hash of σ_e yields a random value R_e. This R_e is used to pseudo-randomly permute the list of validators (with weights by stake). The top of this permutation becomes the leader for the next epoch, and the next k validators become a hot standby committee. The new leader continues the VDF chain from where the previous left off (it has the last output and can keep computing forward). This process ensures an unpredictable but fair rotation of leaders. The use of BLS aggregated randomness (similar to Ethereum's RANDAO or Dfinity's random beacon) provides unbiased randomness that the outgoing leader cannot easily manipulate (especially if combined with a VDF to reveal it after a delay, though here we use immediate signature randomness).

Performance SLA and Slashing: To enforce that leaders actually maintain the 100 us pace and include transactions, we introduce a performance monitoring scheme. Define a target tick rate $\rho^* = 10,000$ tx/s (1 tx per tick on average). Every N ticks (say $N = 10,000$ ticks, which is 10,000 ms = 10 seconds, or some convenient window), the network measures the actual rate ρ_b achieved by the current leader. If the leader is significantly underperforming (producing fewer ticks/txs than expected without good reason), it can be penalized. For example, if the leader's throughput is more than 5% below target (i.e. they slowed the clock or missed many transactions), a portion of their stake is slashed. This creates an incentive to not go idle or deliberately slow down. Additionally, if a leader is caught equivocating (creating two conflicting chains for the same epoch), that is a slashable offense (full stake slash, as it undermines consensus). The standby committee serves as backup: if the leader fails or is slashed, the highest-ranked standby takes over immediately and continues the chain, publishing a notarized handover record linking the last commitment of the old chain to its first new tick. Any attempt by a malicious outgoing leader to interfere with handover (e.g., also publishing a different continuation) results in a provable fork and slashing. This ensures smooth leader transition without loss of continuity.

5.3 Adaptive Difficulty (Hardware Evolution)

Continuum's tick interval Δ (and the corresponding VDF difficulty n sequential steps per tick) is initially set based on current hardware capabilities (e.g. 100 us per tick given 2025-era VDF ASICs). However, hardware speeds will inevitably improve (faster ASICs could compute the VDF faster than 100 us). If left unadjusted, an adversary with a next-generation chip might gain the ability to compute ticks slightly faster and thus subvert the time assumption. To prevent this, Continuum incorporates an adaptive difficulty retargeting algorithm reminiscent of Bitcoin's difficulty adjustment (but on a much shorter time scale and targeting real-time).

The network periodically measures the actual time taken to compute a large number of VDF steps and adjusts n (the number of squarings per tick) to keep the average tick close to 100 us. For example, let W be a window of, say, 2^{32} squarings (which is roughly 1000 s of computation at 100 us per tick). Each node can observe the wall-clock duration of the last W squarings (since we have timestamps for ticks). If the median duration of that window is shorter than expected (meaning hardware got faster and ticks are happening in <100 us), the protocol increases n slightly. If ticks are slower than 100 us (perhaps due to higher security margin or network delay), it decreases n. A simple control algorithm is:

- If observed tick time $< 0.95\Delta$, then $n \leftarrow \lceil 1.05 \cdot n \rceil$ (increase difficulty by 5%)
- If observed tick time $> 1.05\Delta$, then $n \leftarrow \lfloor n/1.05 \rfloor$ (decrease difficulty by $\sim$5%)
- Otherwise, leave n unchanged.

This adjustment might be performed every 1000 seconds or so, gradually adapting to keep $\Delta \approx 1$ ms. The 5% threshold and step-size are tunable parameters to avoid oscillations.

The goal is to ensure no adversary can secretly outrun the clock. If someone invents a faster VDF circuit, the network will notice that ticks are coming quicker and will raise the difficulty, nullifying that advantage. This forces anyone trying to beat the clock to continuously outpace the entire network's difficulty adjustments – essentially impossible if the majority of hardware catches up. In summary, Continuum's security margin evolves in lock-step with hardware advances: if faster chips arrive, just lengthen the chain's work per tick to maintain the 100 us wall-clock interval. This keeps the time oracle honest over the long term without giving up performance gains from hardware improvements for honest participants (everyone will eventually use the faster chips and the system will just handle more squarings per ms).

6 Economic Model and Incentives

Continuum's economic model is designed to minimize extrinsic incentives that could reintroduce unfairness (like priority fees for ordering) while still providing sustainable rewards for sequencers and protecting against spam or denial-of-service.

6.1 Minimal Fee Market and Spam Prevention

Continuum eschews the typical first-price auction or priority gas fee model for transaction inclusion. Since ordering is determined by time and not by fee, users cannot bid to be earlier than others – the best they can do is get their transaction in as soon as possible. However, there still needs to be a mechanism to handle network congestion (when users collectively submit more than 1 transaction per ms on average) and to prevent spam attacks where someone floods the sequencer with transactions.

To achieve this, Continuum uses a single dynamic base fee mechanism inspired by Ethereum's EIP-1559. There is a base fee b (measured, say, in the protocol's native token or USD) that rises and falls with utilization of the 100 us slots. If the transaction throughput is at or below the target (1 tx per tick on average, utilization $u^* = 1.0$ or 100%), the base fee remains low (or near a minimum). If transactions arrive faster than can be processed ($u > 1$), the base fee will start to increase, making it more costly to send transactions and thereby throttling demand.

Concretely, every adjustment window (e.g. every 1000 ticks), the base fee b is updated according to the formula:

$$b \leftarrow b\,(1 + \alpha(u - u^*))\,, \tag{2}$$

where u is the observed average utilization (txs per tick) in that window and α is a small adjustment factor (for example $\alpha = 1/8$ as in Ethereum). If $u > 1$, b goes up; if $u < 1$, b goes down. This mirrors EIP-1559's base fee logic, aiming to target $u = 1$ (full utilization without overload).

Importantly, Continuum's base fee is not used to prioritize transactions within a block (since there are no blocks and no intra-block competition); it is only used as a mechanism to price scarce throughput and prevent abuse. All users pay roughly the same base fee for their transactions, and this fee can be burned or distributed as rewards (see below). There is no advantage in paying more than the base fee except to expedite inclusion when the network is congested – and even then, because ordering is by time, paying a higher fee only helps if it causes the base fee to rise and some other transactions to drop out. There are no per-transaction bidding wars as in traditional gas auctions. In times of congestion, the base fee will naturally rise until demand equals supply (1 tx/ms). Users who value inclusion will tolerate the higher fee; others will wait.

From a spam-resilience perspective, this dynamic fee makes it extremely costly to sustain an attack. If an attacker tries to flood the system with transactions at a rate $r \gg \rho^*$ (much higher than the system can handle), the base fee will quickly skyrocket. For example, suppose the attacker submits 100,000 tx/s (10× the capacity). The base fee would adjust upward, perhaps doubling every small interval, until the cost to keep spamming becomes prohibitive. A simple calculation shows that at 10× overload ($r = 10$ tx per tick), the cost ramps up such that the attacker would be spending on the order of millions of dollars per day to keep spamming. Even a large attacker will run out of money or motivation, meaning volumetric denial-of-service on the ordering layer is economically infeasible without an equal economic weight (which honest users presumably also have, in which case it's just legitimate high usage).

6.2 Incentives for Sequencers and Participants

Continuum's incentive structure must reward the sequencer (or sequencer validators in a committee) for their work, and optionally reward standby nodes that are ready to take over. Since we avoid priority fees, the primary source of revenue is the base fee (which can be thought of as a protocol-level fee, like a toll for using the service). This fee could be burned in part (to benefit all token holders via deflation, similar to Ethereum's model) and/or given to the operators as reward.

One plausible allocation (from the Continuum working design) is: 60% of collected fees go to the active sequencer (leader), 20% go to the standby committee, and 20% go to a treasury for long-term ecosystem funding (e.g. insurance fund, audits, development). This split ensures the leader is well-compensated for running high-performance hardware (ASICs, etc.), but also that standby nodes – who must also run hardware and be online to step in – get some reward for their availability. The treasury cut provides a public good fund and can underwrite any unforeseen expenses (or compensate users in rare events of failure).

To give a rough idea of magnitude: if the base fee bottoms out at, say, \$0.0001 (one-hundredth of a cent) per tx and the system is running at full 10k TPS capacity, the weekly revenue is on the order of \$105,000. This would be split as ∼\$63k to the sequencer, \$21k to standby validators, \$21k to treasury per week under the above allocation. In times of congestion (higher base fee), the revenue (and thus rewards) would be higher, which is appropriate because the sequencer is handling more intensive load and possibly users are willing to pay more. In times of low usage, the base fee might drop near zero, meaning users essentially use the service for free (minus maybe a nominal minimum fee to avoid zero-cost spam).

The protocol can also include slashing penalties and rewards: if a sequencer equivocated and was slashed, some portion of its stake could be redistributed to honest participants or to those who reported the misbehavior. This further incentivizes everyone to monitor the chain (which they can do by simply verifying signatures and hashes).

One more subtle incentive: because Continuum's core design does not reward based on ordering (no MEV extraction for the sequencer if they behave), the sequencer's rational strategy is simply to maximize throughput (to earn more fees by including more transactions) and to keep the system honest (to avoid slashings). This aligns the sequencer's interest with the network's interest – a contrast to traditional block production where miners can profit more by reorderings or selective inclusion.

Finally, the standby committee incentive (the 20% share) ensures that multiple validators remain online and synchronized with the VDF, ready to take over. They earn this share essentially as a high-availability reward. If the leader fails, a standby that takes over will then start getting the leader reward portion, so there is a natural rotation and competition.

Overall, Continuum's economic model strives to create a sustainable, fair sequencing service: users pay a minimal fee that only rises during congestion (and even then does not distort ordering), sequencers are paid for providing a reliable clock and ordering service, and malicious behavior is discouraged through cryptographic guarantees and financial penalties. By combining cryptographic fairness with economic incentives, Continuum aims to be a credibly neutral but economically secure base layer for ordering.

7 Conclusion

We presented Continuum, a novel blockchain transaction sequencing protocol that eliminates the concept of blocks in favor of a continuous-time log with cryptographically enforced ordering. Continuum relies on a verifiable delay function to act as a decentralized clock, producing a tick every 100 us and tying each transaction to a specific time. We defined the protocol's mechanics in detail, provided a formal model of its state transition rules and validity conditions, and proved key security properties including order integrity (no timestamp inversions) and non-equivocation (a single canonical history). We analyzed the system's liveness and performance, showing that even a single sequencer node can achieve throughput on the order of 10,000 TPS with sub-second global finality – far beyond traditional blockchain performance, and with fairness guarantees that tackle MEV at its root.

Continuum can be seen as an embodiment of time-based consensus: rather than block proposers competing or colluding with free rein over ordering, time itself (instantiated via a sequential VDF) is the arbiter of order. By anchoring every transaction to a time tick that cannot be forged or hurried, Continuum ensures fairness and consistency in a way previously only theorized in academic contexts like order-fair Byzantine consensus protocols [12,13]. Our security analysis indicates that basic properties like order integrity and finality hold under standard cryptographic assumptions (and even a determined adversary with slightly faster hardware would be caught or neutralized by difficulty adjustments). The performance evaluation and design parameters suggest that Continuum's approach is practical: modern hardware and networks can support its requirements, and the adaptive mechanism keeps it future-proof.

We also discussed how Continuum could be integrated across the blockchain ecosystem – from serving as a fair sequencing layer for rollups and DeFi applications to coordinating cross-chain operations. By providing a transparent and verifiable global timeline, Continuum could act as a shared sequencing service, a public utility that many protocols tap into for ordering guarantees. Advanced extensions like the commit-reveal scheme further enhance fairness by blindening transaction contents during ordering, and a proof-of-stake based rotating sequencer model brings decentralization and robustness to the system.

Continuum offers a modular solution for transaction ordering: continuous, fair, and fast. It combines cryptographic time proofs with blockchain consensus to yield a system where time is the final arbiter of truth. We believe this model can significantly reduce MEV and improve the user experience and security of decentralized applications. Future work will focus on optimizing the VDF computation and verification, and deploying it in real-world scenarios such as layer-2 networks or as a standalone high-throughput ledger. We invite the community to explore this direction, as provably fair and instant transaction ordering could become a cornerstone of the next generation of blockchain scalability and interoperability.

References

[1] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *CRYPTO 2018: Advances in Cryptology*, volume 10991, pages 757–788, 2018.

[2] Flashbots. MEV and the Limits of Scaling. Flashbots Writings, 2025. Available: `https://writings.flashbots.net/mev-and-the-limits-of-scaling`.

[3] V. Buterin. An incomplete guide to rollups. Vitalik's Blog, Jan. 2021. Available: `vitalik.ca/general/2021/01/05/rollup.html`.

[4] I. Cascudo, B. David, O. Shlomovits, and D. Varlakov. Cornucopia: Distributed randomness beacons at scale. In *5th Conference on Advances in Financial Technologies (AFT 2024)*, LIPIcs vol. 316, pages 17:1–17:25, 2024.

[5] Chainlink Labs. Fair sequencing services: Enabling a provably fair defi ecosystem. Chainlink 2.0 Whitepaper, 2021.

[6] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. In *IEEE Symposium on Security and Privacy (SP)*, pages 910–927, 2020.

[7] Ethereum Foundation. Ethereum 2.0 phase 0 – the beacon chain. Ethereum 2.0 Specification, 2020. Available: `github.com/ethereum/consensus-specs`.

[8] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. Continuous verifiable delay functions. In *EUROCRYPT 2019*, pages 125–154. Springer, 2019. Also available as IACR ePrint Report 2019/619.

[9] Flashbots Research. A transaction ordering rules taxonomy. Flashbots Research, Jan. 2023. Available: `writings.flashbots.net/order-flow-auctions-and-centralization`.

[10] A. Juels. Fair sequencing services: Enabling a provably fair defi ecosystem. Chainlink Blog, Sept. 2020. Available: `chain.link` blog.

[11] A. Kiayias, S. Quader, and A. Russell. Consistency of proof-of-stake blockchains with concurrent honest slot leaders. IACR ePrint Archive, Report 2020/041, 2020. Available: `eprint.iacr.org/2020/041`.

[12] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels. Order-fairness for byzantine consensus. IACR Cryptology ePrint Archive, Report 2020/269, 2020.

[13] V. Nikolaenko, S. Kelkar, D. Malkhi, and T. Rabin. Order-fairness for byzantine consensus. Cryptology ePrint Archive, Report 2019/862, 2019. Available: `eprint.iacr.org/2019/862`.

[14] M. S. Öztürk. Modular multiplication algorithm suitable for low-latency circuit implementations. In *SNUG 2020: Synopsys Users Group Conference*, VDF Alliance, 2020.

[15] L. Rotem. Simple and efficient batch verification techniques for verifiable delay functions. IACR ePrint Archive, Report 2021/1209, 2021. Available: `eprint.iacr.org/2021/1209`.

[16] B. Wesolowski. Efficient verifiable delay functions. In *EUROCRYPT 2019: Theory and Applications of Cryptographic Techniques*, pages 379–407. Springer, 2019.

[17] B. Wesolowski. Efficient verifiable delay functions. *Journal of Cryptology*, 33(4):2113–2147, 2020. Available: `ir.cwi.nl/pub/30021/`.

[18] A. Yakovenko. Solana: A new architecture for a high performance blockchain. Whitepaper v0.8.13, 2018. Available: `solana.com/solana-whitepaper.pdf`.