Chain of Randomness - An attempt to prevent MEV

Frederik Alexander, Julian Winnetou Sosa, Eden Baudin

Abstract

We propose a consensus-layer mechanism to dramatically reduce extractable MEV on Ethereum by **forcing block content to be the output of a public, verifiable random sampling process over the mempool**, rather than an arbitrary ordering chosen by builders. The core idea is: (i) validators/builders *commit* to their mempool snapshot; (ii) a public randomness source (Ethereum's beacon-chain RANDAO exposed via `PREVRANDAO`) seeds a two-stage sampling procedure that deterministically selects a small candidate set and then a final *fixed-size* block set; (iii) the final intra-block order is a verifiable random permutation derived from the same seed(s). Peers audit the builder's commit with a lightweight set-reconciliation sketch and a membership-proof spot-check; blocks that deviate are invalid. Under reasonable mempool sizes, the probability that an attacker can place both a target transaction and their frontrun in the *same block* in the *right order* becomes negligible (back-of-the-envelope numbers below), while bandwidth and latency overheads remain small.

1 Idea Description

- **Consensus rule-change (via hard fork):** A block is valid only if its transactions are the result of an on-chain verifiable sampling-and-shuffle procedure over the (builder/validator's) mempool snapshot taken at a specified slot boundary.
  - *Commit:* Before building, the builder publishes a *commitment* to its mempool snapshot (e.g., a Merkle root over the sorted list of txids).
  - *Audit sketch:* Alongside the commitment, the builder gossips a compact *set-reconciliation sketch* (e.g., MinHash signatures + a small IBLT/Graphene-style structure) so peers can estimate $\geq 80\%$ overlap with their own mempool without exchanging the full set.
- **Public randomness:** Use the beacon chain's RANDAO entropy (accessible in the EVM as `PREVRANDAO` per EIP-4399) to seed sampling and the final permutation. This removes builder discretion in content and ordering.
- **Two-stage sampling pipeline:**
  - *Stage A:* From (say) $N = 100{,}000$ mempool transactions, derive a *candidate set of 100* via seed S_1.
  - *Stage B:* From that 100, derive the *final block set of 30* via seed S_2 (S_2 can be derived independently from RANDAO or from S_1 with domain separation).
  - *Random intra-block order:* Apply a seed-derived permutation to the 30 before sealing the block.
  - *Parameter note:* the numbers 100k$\rightarrow$100$\rightarrow$30 are illustrative; in deployment they should match gas-limit and target throughput.

- **Enforcement:**
  - Peers verify: (i) the mempool commit; (ii) membership proofs for a small random audit sample; (iii) that the block's txids equal the seed-selected set and the order equals the seed-derived permutation.
  - Blocks that deviate are invalid under fork-choice; repeated misbehavior is slashable.

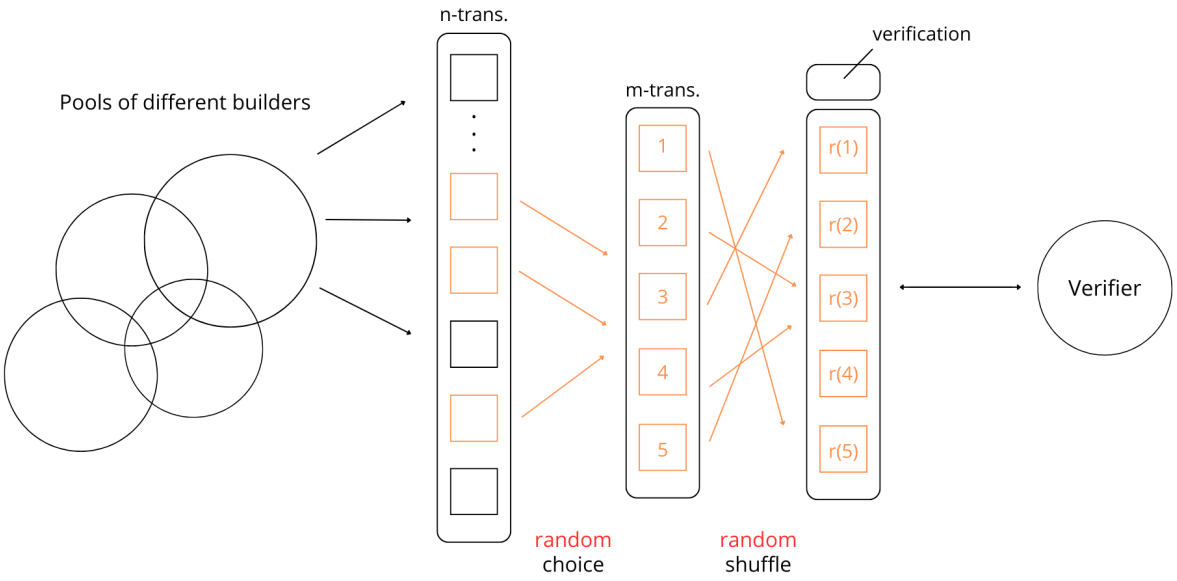

Figure 1: Diagrammatic View of the idea

2 80% Consensus on the Mempool Snapshot

- **Goal:** Ensure the builder's mempool view is not an outlier engineered to include/exclude specific transactions.
- **Mechanism:**
  - The builder publishes: (a) a commitment (root) to its mempool txid multiset; (b) a compact sketch to allow *similarity estimation* without bulk transfer.
  - Peers compute an estimated *Jaccard similarity* $J = \frac{|A \cap B|}{|A \cup B|}$ between their mempool A and the builder's B using *MinHash*, and, if needed, reconcile small differences using *IBLT/Graphene*-style round-trips. Target: $J \geq 0.8$ ($\geq 80\%$ match).
  - **Bandwidth-efficient audits:** Instead of shipping 100k txids, we exchange k MinHash signatures ($k \ll N$) and a small IBLT to identify a few discrepancies; *membership proofs* for a random spot-check subset (dozens to hundreds) provide cryptographic assurance the committed set matches the sketch.

- **Consensus hook:** If the similarity falls below 80% (or if too many audit proofs fail) the block is *invalid*. This ties liveness to being in-step with the *honest* network mempool, discouraging selective view-crafting and shadow mempools.
- **Why 80%?** It's a tunable threshold balancing: (i) natural mempool drift and network latency vs. (ii) the need to prevent targeted curation. Empirically calibrated thresholds can be refined in testnets.

3 Shuffling

To prevent brute-forcing of possible preorders (so the random shuffle produces a good result), we propose a "slow shuffle" that cannot be brute-forced.

Slow seeded shuffle (conceptual, with modular squaring).

Work in the finite field $\mathbb{F}_p = \mathbb{Z}/p\mathbb{Z}$ for a large prime p. For any integer x, write

$$[x]_p \coloneqq x \bmod p \in \{0, \dots, p-1\}.$$

From a seed-derived pseudorandom stream (x_i), define for each position i a "heavy" key by iterating the square-and-add map

$$h : \mathbb{F}_p \to \mathbb{F}_p, \qquad h(z) \equiv z^2 + z \pmod{p} \quad (\text{equivalently } h(z) = [\, z^2 + z \,]_p = [\, z(z+1) \,]_p).$$

Initialize and iterate:

$$y_0 \coloneqq [x_{i-1}]_p, \qquad y_{t+1} \coloneqq h(y_t) \quad (t \geq 0), \qquad k_i \coloneqq y_R.$$

Finally, pair (k_i, a_i), perform a stable insertion sort by k_i, and output the a_i in that order. The complexity grows as O(nP), so this achieves our result.

4 Probability Calculations

Assume the final block contains n transactions sampled *uniformly at random without replacement* from a mempool of size N.

- **Victim V and attacker A both included.**

$$\Pr[\{V, A\} \subseteq \text{block}] = \frac{\binom{N-2}{n-2}}{\binom{N}{n}} = \frac{n(n-1)}{N(N-1)}.$$

- **Front-run order (attacker before victim).** Conditioned on both being included, their relative order is uniform:

$$\Pr[A \prec V] = \tfrac{1}{2} \, \frac{n(n-1)}{N(N-1)}.$$

- **Attacker spams m distinct frontruns.** Let $p_{\text{fr}} \coloneqq \frac{1}{2} \, \frac{n(n-1)}{N(N-1)}$.

$$\Pr[\text{any succeeds}] \approx 1 - (1 - p_{\text{fr}})^m \approx m \, p_{\text{fr}} \quad (\text{for small } p_{\text{fr}}).$$

- **Sandwich (one attacker pair A_1, A_2 around V).** All three must be included, and the order must be A_1, V and then A_2, with maybe others in between.

$$p_{\text{sand}} \;\coloneqq\; \Pr[\text{sandwich with one pair}] \;=\; \frac{1}{6} \, \frac{n(n-1)(n-2)}{N(N-1)(N-2)}.$$

Example Calculations ($n = 30$).

	$N = 100{,}000$	$N = 1{,}000{,}000$
$\Pr[\{V, A\} \subseteq \text{block}]$	8.70×10^{-8}	8.70×10^{-10}
$\Pr[A \prec V]$	4.35×10^{-8}	4.35×10^{-10}
Pr[FR], m=100	4.35×10^{-6}	4.35×10^{-8}
Pr[FR], m=1000	4.35×10^{-5}	4.35×10^{-7}
Pr[SW]	4.06×10^{-12}	4.06×10^{-15}

m: number of front-run attempts.

Notes.

- The two-stage $N \to 100 \to 30$ pipeline that is uniform at each stage is distributionally equivalent to sampling 30 directly from N.
- Calibrate N and n to deployment realities; these values are illustrative.

5 Literature Review

There are different approaches to prevent MEV.

One approach to mitigating MEV is to make transaction ordering unpredictable using verifiable randomness. The Helix protocol [1] exemplifies this strategy by integrating a random beacon into the block proposal process. In Helix, transactions are ordered according to a cryptographic lottery, so no miner or validator can bias which transaction comes first. This means even if a participant wanted to reorder transactions for profit, the random ordering would prevent it. This idea closely aligns with our Chain of Randomness proposal, which also uses randomness to neutralize any single actor's control over ordering. The key difference is that Helix is a custom consensus protocol built around random ordering, whereas Chain of Randomness aims to add a verifiable random ordering mechanism on top of existing blockchain infrastructure without overhauling the entire consensus.

Another relevant line of research combines randomness with transaction hiding to curb MEV. Piet et al.'s MEVade system [2] is a recent example. In MEVade, transactions in each block are kept hidden (encrypted) and then shuffled using a public random beacon (e.g. Ethereum's RANDAO) before execution. Because the order is determined by an unpredictable, verifiable random value – and only revealed at execution time – even a block producer cannot preferentially position their own transactions. This approach relates to Chain of Randomness in that both leverage unpredictable randomness to ensure fair ordering of transactions. However, MEVade goes further by requiring heavy cryptographic steps (encrypting transactions and collective shuffling), while our proposal focuses on achieving fairness with a lighter-weight random ordering of visible transactions. In short, MEVade's random permutation of encrypted transactions guaranties fairness but at the cost of added complexity, while Chain of Randomness seeks a simpler way to introduce randomness into the ordering process.

A complementary direction is improving the auditability of the mempool to hold block producers accountable. The LØ protocol introduced by Nasrulin et al. [3] takes this approach by creating an accountable mempool. In LØ, each miner/validator maintains a secure log of all pending transactions it has received – essentially a timestamped mempool snapshot that is shared or committed to other nodes. This record allows later to verify exactly which transactions were available before block creation. If a miner omits or reorders transactions unfairly (for example, by dropping

a user's transaction to insert their own), the discrepancy will be detected by comparing publicly committed mempool snapshots. This auditability concept ties in with the second aspect of Chain of Randomness: ensuring that the state of the mempool can be verified by everyone. Both LØ and our proposal aim to prevent hidden manipulations, but they do so differently. LØ focuses on detecting and blaming any unethical ordering after the fact – miners must "play by the rules" or be exposed – while Chain of Randomness aims to prevent the opportunity for manipulation in the first place by randomizing order. In our solution, the auditable mempool snapshots would complement the random ordering by providing transparency: the random selection for ordering is performed on a known set of transactions, so any attempt to cheat (by sneaking in or excluding transactions) would be apparent.

References

[1] D. Yakira et al., "Helix: A fair blockchain consensus protocol resistant to ordering manipulation," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 1584–1597, 2021.

[2] J. Piet, V. Nair, and S. Subramanian, "MEVade: An MEV-resistant blockchain design," in *Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, IEEE, 2023, pp. 1–9.

[3] B. Nasrulin et al., *LØ: An accountable mempool for MEV resistance*, 2023. arXiv: 2307.02081.