Internet-Draft
Intended status: Experimental
Expires: September 15, 2026

Heime Jorgen
Kenosian

March 15, 2026

The TLS TimeToken Secure Protocol (tttps://) draft-helmprotocol-tttps-00

Abstract

This document specifies the TLS TimeToken Secure Protocol (tttps://), an application-layer protocol extension that augments TLS 1.3 [RFC8446] with cryptographically verifiable temporal ordering. Where TLS authenticates identity ("who"), tttps:// authenticates temporal origin ("when") in a manner resistant to Byzantine manipulation.

The protocol introduces Proof of Time (PoT): a multi-source NTP-synthesized, cryptographically authenticated timestamp bound to a distributed ledger anchor. PoT enables honest participants to achieve preferential execution, while Byzantine nodes are economically evicted through an adaptive state machine (the AdaptiveSwitch mechanism).

Primary use cases include MEV-resistant decentralized exchange (DEX) transaction ordering, AI agent-to-agent payment sequencing, IoT mission-critical command ordering, and satellite inter-link communication timestamping.

This document has Experimental status to allow deployment experience to accumulate prior to consideration for the Standards Track. The GRG Integrity Pipeline specification will be made available for independent implementation upon conclusion of pending patent proceedings (see Section 10).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 15, 2026.

This document is being discussed in the context of a proposed Birds of a Feather (BoF) session to explore the formation of a new working group on temporal ordering for distributed systems. Interested parties are invited to review the draft and participate. Comments are welcome on the dispatch@ietf.org mailing list.

Copyright Notice

Table of Contents

1. Introduction

1.1. Motivation

The Internet operates under two implicit trust axioms:

- Identity trust (solved by TLS/PKI): "Who am I communicating with?"
- Temporal trust (currently unsolved at the protocol level): "When did this message originate, in a manner that cannot be falsified by any single party?"

Emerging application domains -- decentralized exchanges, autonomous AI agent networks, industrial IoT systems, and inter-satellite

communication links -- share a common requirement: verifiable temporal ordering of events across mutually untrusted parties. These environments were not addressed by existing time synchronization or trust infrastructure, which were designed primarily for single-domain or human-supervised contexts.

This document describes an experimental protocol for a temporal trust layer leveraging distributed ledgers. The proposed mechanism provides a transparent and verifiable framework for time-ordered event sequencing in multi-party, adversarial environments. It is designed to operate alongside existing trust infrastructure, not as a replacement for it.

Block builder timestamp manipulation enables MEV sandwich attacks [FLASHBOTS] costing an estimated $60 million annually in DEX user losses as of 2025. GPS spoofing enables coordinate falsification in autonomous systems. NTP MITM attacks allow timestamp manipulation in financial systems.

This specification addresses temporal Byzantine faults: network participants that report plausible but fabricated timestamps to gain economic advantage or disrupt system operation.

1.2. Scope

This document specifies:

- The Proof of Time (PoT) data structure and generation algorithm
- The GRG Integrity Pipeline abstract interface (implementation details are outside the scope of this document)
- The AdaptiveSwitch Byzantine eviction mechanism
- The tttps:// TLS extension for embedding PoT in TLS 1.3 handshakes

This document does NOT specify:

- Concrete implementation of the GRG pipeline cryptographic operations (covered by pending patent applications; see Section 10)
- Specific NTP server selection policies
- Smart contract implementations for on-chain anchoring
- Pricing or fee schedules (implementation-defined; see Section 7)

1.3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174].

Proof of Time (PoT):
: A cryptographically authenticated record of a synthesized timestamp derived from multiple independent NTP sources, bound to a context identifier and protected against replay.

GRG Pipeline:
: A multi-stage integrity pipeline that encodes and authenticates PoT payloads. "GRG" denotes the three pipeline stages: G(olomb-Rice encoding), R(eed-Solomon erasure coding), and G(olay forward error correction), with keyed integrity

authentication applied at each stage. The abstract interface is specified in Section 4; concrete implementation details are proprietary and covered by pending patents (Section 10).

AdaptiveSwitch:
A state machine that classifies network participants as TURBO (honest, preferential execution) or FULL (potentially Byzantine, standard execution) based on observed PoT conformance.

Byzantine Time Attack:
An adversarial action in which a network participant reports a fabricated or manipulated timestamp to gain ordering advantage.

Temporal Ordering:
The verifiable sequence in which transactions or messages were initiated, independent of network propagation delays.

PoT Issuer:
An entity authorized to generate and sign Proofs of Time for consumption by verifying parties. Analogous in function to a Certificate Authority in the TLS PKI model, but operating over time attestations rather than identity certificates.

2. Problem Statement: Temporal Indeterminacy

2.1. The Byzantine Time Problem

In current decentralized systems, "time" is approximated. Block timestamps in Ethereum are provided by block builders with second-level granularity and no cryptographic commitment to external time sources. This creates a "dark window" within which transactions may be reordered, front-run, or censored.

Formally, let T_actual be the actual transaction submission time and T_block be the block timestamp assigned by the builder. Current systems provide no mechanism to verify that:

$$|T_{block} - T_{actual}| < \epsilon$$

for any meaningful epsilon. A malicious builder may set T_block arbitrarily within consensus tolerance, enabling sandwich attacks [EIGENPHI], front-running, and other temporal manipulation.

The same vulnerability exists in:

- AI agent multi-step transactions where ordering determines economic outcome
- Industrial IoT command sequences where ordering determines physical safety
- Inter-satellite communication where propagation delay variance is exploited

2.2. Existing Mitigations and Their Limitations

Private mempools (e.g., Flashbots Protect [FLASHBOTS-PROTECT]) reduce front-running exposure but do not eliminate it. As documented in [SANDWICH-STUDY], private sandwich attacks affecting 3,126 transactions produced $409,236 in losses during November-December 2024, with 50.1% of Ethereum transactions using private routing by February 2025. Private routing does not provide temporal authentication; it only obscures order flow.

Network Time Security (NTS) [RFC8915] addresses NTP

authentication within a single network domain but does not provide:

- Cross-domain Byzantine fault tolerance
- Economic incentive alignment for honest participation
- Integration with distributed ledger anchoring

IEEE 1588 Precision Time Protocol (PTP) provides sub-microsecond synchronization within local networks but is not Byzantine fault tolerant and does not operate across administrative domains.

3. Proof of Time (PoT) Specification

3.1. PoT Data Structure

A Proof of Time is defined as the following JSON-serializable structure. Implementations MUST preserve all fields for verification purposes.

```
struct ProofOfTime {
    uint64  timestamp_ns;
        // Synthesized Unix timestamp in nanoseconds.
        // Derived from multi-source NTP median synthesis
        // per the algorithm in Section 3.2.

    uint64  expires_at_ms;
        // Validity window expiry in Unix milliseconds.
        // Default: timestamp_ns / 1,000,000 + 60,000.
        // Implementations MUST reject PoTs where
        // current_time_ms > expires_at_ms.

    uint8   sources;
        // Number of independent NTP sources contributing
        // to synthesis.  MUST be >= 2 for TURBO eligibility.
        // A PoT with sources == 1 is valid but carries
        // degraded confidence and is ineligible for TURBO.

    uint8   stratum;
        // Minimum NTP stratum across contributing sources.
        // Lower values indicate proximity to atomic clocks.
        // MUST be <= 15 (RFC 5905 Section 7.3).

    uint32  confidence_ppm;
        // Source availability confidence scaled by 10^6.
        // Value = (responding_sources / configured_sources)
        //         * 1,000,000.
        // MUST be >= 500,000 (0.5 confidence) for
        // TURBO eligibility.

    bytes32 nonce_hash;
        // keccak256(random_nonce_bytes).
        // Provides replay protection.  The underlying
        // nonce MUST be generated using a cryptographically
        // secure random number generator.

    bytes32 grg_hash;
        // Output of the GRG Integrity Pipeline applied to
        // the PoT payload.  See Section 4.3.

    bytes   issuer_signature;
        // Optional Ed25519 signature over on_chain_hash
        // by the PoT issuer.  Enables non-repudiation.

    SourceReading[] source_readings;
```

```
        // Per-source evidence supporting timestamp_ns.
    }

    struct SourceReading {
        string  source_name;    // Identifier of the time source
        uint64  timestamp_ns;   // Per-source timestamp
        float32 uncertainty_ms; // Per-source uncertainty (ms)
        uint8   stratum;        // Per-source NTP stratum
    }
```

3.2. PoT Generation Algorithm

A PoT is generated as follows:

(1) Query N configured time sources in parallel. Sources MAY include NTPv4 [RFC5905] servers and HTTPS-authenticated endpoints. Implementations SHOULD use a minimum of 3 sources.

(2) Discard responses where:
- stratum == 0 or stratum > 15
- Round-trip delay exceeds implementation-defined threshold
- Response timestamp is prior to 1970-01-01 (Unix epoch)

(3) For each valid response i, compute:

$$\theta_i = ((T2_i - T1_i) + (T3_i - T4_i)) / 2$$

where T1 is local originate time, T2 is server receive time, T3 is server transmit time, and T4 is local receive time, all in nanoseconds.

$$timestamp_i = T4_i + \theta_i$$

(4) Sort valid timestamps: $t_1 \le t_2 \le \ldots \le t_k$.

(5) Select synthesized timestamp:
- If k == 1: $T_{synth} = t_1$ (with degraded confidence warning)
- If k == 2: $T_{synth} = (t_1 + t_2) / 2$
- If k >= 3: $T_{synth} = t_{\{floor(k/2)\}}$ (median)

(6) Verify all source readings are within tolerance of T_{synth}:

$$|t_i - T_{synth}| \le TOLERANCE(stratum)$$

where TOLERANCE is defined in Section 5.2.

(7) Construct PoT with T_{synth}, a fresh cryptographic nonce, and expiry window. Apply GRG pipeline (Section 4) to produce grg_hash.

3.3. On-Chain Hash (Deterministic ABI Encoding)

The on-chain representation of a PoT MUST use the following deterministic encoding to enable independent third-party verification:

```
    on_chain_hash = keccak256(
        ABI.encode(
            uint64(timestamp_ns),
            uint64(expires_at_ms),
            uint8(sources),
            uint8(stratum),
            uint32(confidence_ppm),
            bytes32(keccak256(nonce_bytes))
        )
    )
```

This encoding is field-order-fixed and independent of serialization library implementation, enabling any conforming implementation to reproduce the hash given the same PoT fields.

NOTE: JSON serialization MUST NOT be used for on_chain_hash computation, as key ordering varies across implementations.

3.4. PoT Verification

A verifying party MUST reject a PoT if any of the following conditions hold:

(a) current_time_ms > expires_at_ms

(b) confidence_ppm < 500,000

(c) sources < 1 (a PoT with zero sources is invalid by definition; sources == 1 is valid but TURBO-ineligible per Section 3.1)

(d) stratum > 15

(e) The nonce has been previously observed (replay protection)

(f) grg_hash fails GRG pipeline verification (Section 4.3)

(g) Any source_reading deviates from timestamp_ns by more than TOLERANCE(stratum) nanoseconds, where TOLERANCE is:
- stratum <= 1: 10,000,000 ns (10 ms)
- stratum == 2: 25,000,000 ns (25 ms)
- stratum >= 3: 50,000,000 ns (50 ms)

A PoT with sources < 2 or confidence_ppm < 750,000 MUST NOT be admitted to TURBO mode regardless of other fields.

Optionally, verifying parties SHOULD validate issuer_signature against a trusted PoT issuer public key registry.

4. GRG Integrity Pipeline (Abstract Specification)

4.1. Pipeline Overview

The GRG Integrity Pipeline is a multi-stage encoding and authentication mechanism applied to PoT payloads. It provides:

- Structured encoding for error correction (Stage 1)
- Erasure coding with configurable redundancy (Stage 2)
- Forward error correction per shard (Stage 3)
- Keyed integrity authentication per shard (Stage 4)

The pipeline accepts an arbitrary byte sequence and produces a set of authenticated shards. Any conforming subset of shards can reconstruct the original payload and verify its integrity.

The specific algorithm parameters, optimization strategies, and implementation techniques are proprietary and covered by pending patent applications (see Section 10). This specification defines only the abstract interface required for interoperability.

4.2. Stage Descriptions

Stage 1 -- Structured Encoding (G):
The input payload is transformed into a structured bit representation using a Golomb-Rice based encoding optimized for

subsequent error correction. Implementations MUST preserve the original byte length as metadata to enable exact reconstruction.

Stage 2 -- Erasure Coding (R):
The structured payload is divided into K data shards and M parity shards (K + M total), using a systematic Reed-Solomon coding scheme over a finite field. Any K of the K + M shards MUST be sufficient to reconstruct the original payload. Default parameters: K=4, M=2 (total 6 shards; recovers from any 2 shard failures).

Stage 3 -- Forward Error Correction (G):
Each individual shard is protected with a Golay-based binary block code capable of correcting up to t bit errors per codeword, where t >= 3.

Stage 4 -- Keyed Integrity Authentication:
Each shard is appended with a truncated keyed hash (minimum 8 bytes) computed with a context-derived key. The key MUST be derived from the deployment context (e.g., chain identifier and pool address) to prevent cross-context replay. The specific key derivation function is implementation-defined; implementations MUST ensure that shards produced for one context cannot be used to satisfy verification for a different context.

4.3. Verification Interface

The GRG pipeline MUST expose the following abstract interface:

```
Input:   original_data (bytes), shards (list of byte arrays),
         context_id (opaque identifier)
Output:  VerifyResult { ok: bool, hmac_failures: int,
                        recovered: bool }
```

A result with ok=true and recovered=true indicates that the payload was successfully reconstructed despite one or more shard integrity failures, using the erasure coding redundancy of Stage 2. Implementations SHOULD log hmac_failures for security monitoring.

The grg_hash field in the PoT structure is computed as:

```
grg_hash = keccak256(concatenate(shards))
```

where shards are the output of Stage 4.

5. AdaptiveSwitch: Byzantine Node Eviction Mechanism

5.1. State Machine

The AdaptiveSwitch classifies each network participant as one of two operational states:

```
TURBO: Preferential execution mode.
       Latency target: 50 ms.
       Fee discount: 20% (implementation-defined).

FULL:  Standard execution mode.
       Latency target: 127 ms.
       No fee discount.
```

All participants begin in FULL state. Transitions are determined by the observed PoT conformance rate over a sliding window.

5.2. Transition Conditions and Hysteresis

Let R_match be the fraction of recent blocks whose observed transaction order and timestamp are consistent with the participant's submitted PoT records, computed over a sliding window W (default W=20 blocks).

Tier-specific temporal tolerance:

```
T0_epoch (6.4 min intervals): 2000 ms
T1_block (2 s intervals):      200 ms
T2_slot  (12 s intervals):     500 ms
T3_micro (100 ms intervals):    10 ms
```

Transitions:

```
FULL -> TURBO: R_match >= 0.95 over W blocks AND
               penalty_cooldown == 0 AND
               GRG integrity verified for all W blocks

TURBO -> FULL: R_match < 0.85 OR
               any GRG integrity failure detected
```

The hysteresis gap (0.95 entry vs. 0.85 maintenance) prevents oscillation and ensures stable TURBO participation requires sustained honest behavior.

5.3. Penalty and Exponential Backoff

Upon a GRG integrity failure while in TURBO state, a mandatory cooldown period is applied:

```
cooldown_blocks = 20 * 2^(consecutive_failures - 1)
```

subject to a maximum of 320 blocks, with consecutive_failures capped at 4.

This exponential backoff ensures that repeated Byzantine behavior results in extended exclusion from TURBO mode, creating a strong economic disincentive for timestamp manipulation.

5.4. Nash Equilibrium Analysis

Let R be the block reward, V_mev be the extractable MEV value, C_turbo and C_full be the per-block costs in TURBO and FULL modes respectively, and L_penalty be the expected penalty cost.

```
U_honest   = R - C_turbo
U_byzantine = R + V_mev - C_full - L_penalty
```

The protocol parameters are chosen such that:

```
U_honest > U_byzantine
```

Specifically, the fee discount in TURBO mode combined with the exponential backoff penalty ensures that sustained honest participation yields strictly higher utility than occasional MEV extraction, provided V_mev does not exceed the penalty cost. Specific parameter calibration is implementation-defined.

6. tttps:// Protocol Integration

6.1. TLS Extension Definition

This document requests the allocation of a new TLS extension type value from IANA:

ExtensionType: pot_temporal_attestation (TBD)

The extension carries a serialized ProofOfTime structure as defined in Section 3.1.

```
struct {
    opaque pot_data<1..2^16-1>;
} TemporalAttestationExtension;
```

where pot_data is the canonical JSON encoding of a ProofOfTime with BigInt fields represented as decimal strings.

6.2. Handshake Augmentation

The tttps:// handshake augments TLS 1.3 as follows:

ClientHello:
MAY include a pot_temporal_attestation extension carrying the client's current PoT.

ServerHello (or EncryptedExtensions):
If the server supports tttps://, it MUST include a pot_temporal_attestation extension carrying the server's current PoT.

Verification:
After the TLS handshake completes, each party MUST verify the counterpart's PoT per Section 3.4. Additionally:

```
|client.timestamp_ns - server.timestamp_ns| <=
    max(TOLERANCE(client.stratum), TOLERANCE(server.stratum))
```

If this condition is violated, the connection MAY be terminated with a new alert: temporal_inconsistency(TBD).

6.3. Backward Compatibility

If either party does not include the pot_temporal_attestation extension, the connection proceeds as standard TLS 1.3 without temporal attestation. This ensures complete backward compatibility. Applications MAY require tttps:// as a policy matter; this document does not mandate such requirements.

7. Tier Structure

tttps:// defines four operational tiers, balancing temporal resolution against verification overhead. The tier determines the temporal tolerance window (Section 5.2) and eligibility for TURBO mode:

Tier	Interval	Resolution Class	Primary Use Case
T0_epoch	6.4 min	low	Batch / standard DEX
T1_block	2 s	medium	L2 block ordering
T2_slot	12 s	high	L1 slot ordering
T3_micro	100 ms	ultra	Institutional / HFT

NOTE: Per-tick pricing and fee schedules are implementation-defined and outside the scope of this specification. Reference pricing for the OpenTTT implementation is documented separately [OPENTTT].

Implementations operating at T3_micro SHOULD employ asynchronous GRG processing to meet the 100 ms interval constraint.

8. IANA Considerations

This document requests the following IANA actions:

1. Registration of TLS extension type pot_temporal_attestation in the "TLS ExtensionType Values" registry.

2. Registration of TLS alert temporal_inconsistency in the "TLS Alerts" registry.

3. Creation of a new "PoT Source Type" registry with the following initial values: ntp_udp, https_date_header, roughtime.

9. Security Considerations

9.1. NTP MITM Attacks

Plain NTPv4 [RFC5905] is transmitted over unauthenticated UDP, making it vulnerable to on-path attackers. Implementations SHOULD use Network Time Security (NTS) [RFC8915] for at least one configured source. Additionally, including HTTPS-authenticated sources provides TLS-protected temporal evidence even at reduced precision (1-second resolution from HTTP Date headers).

The multi-source median synthesis algorithm is inherently resistant to single-source compromise: an attacker must compromise a majority of configured sources simultaneously to meaningfully affect the synthesized timestamp.

9.2. Replay Attacks

The nonce_hash field MUST be generated fresh for each PoT and MUST NOT be reused. Verifying implementations MUST maintain a bounded nonce cache with TTL >= expires_at_ms to reject replayed PoTs. Recommended cache parameters: capacity 10,000 entries, TTL 300 seconds (5x the default 60-second expiry window).

9.3. Sybil Time Sources

A single entity controlling multiple NTP servers could fabricate a consistent false timestamp across "multiple" sources. Implementations SHOULD use sources from distinct administrative domains (e.g., NIST, KRISS, Google, Cloudflare) with verified different operators.

9.4. Side-Channel Considerations

The key derivation function for the GRG pipeline MUST be implemented using constant-time operations where applicable to prevent timing side-channel attacks. The specific derivation algorithm is implementation-defined (see Section 10).

9.5. Byzantine Node Economic Attacks

Section 5.4 demonstrates that the AdaptiveSwitch mechanism creates a Nash equilibrium favoring honest participation. However, implementations MUST set penalty parameters such that L_penalty exceeds the expected V_mev for the deployed context. Operators SHOULD monitor the TURBO/FULL ratio across participants and investigate sustained FULL operation as a potential indicator of Byzantine behavior.

10. Intellectual Property Rights

The contributor represents that to the best of the contributor's knowledge, there are IPR claims under which RAND licensing terms would be available, in the event the IETF takes any action based on the contribution (pursuant to BCP 79 [RFC8179]).

Specifically, pending patent applications cover aspects of the technologies described in this document, including:

(a) The multi-stage GRG integrity pipeline (Stage 1 through Stage 4 as described in Section 4), including specific algorithm parameter selection, key derivation methodology, and performance optimization techniques;

(b) The AdaptiveSwitch Byzantine node eviction mechanism, including the specific hysteresis thresholds, penalty calculation formula, and Nash equilibrium parameterization;

(c) The combination of multi-source NTP synthesis with on-chain deterministic hash anchoring as specified in Section 3.3.

Patent applications have been filed in multiple jurisdictions. Application numbers are not yet publicly available pending examination. The anticipated licensing framework is:

- Open-source non-commercial implementations: royalty-free
- Entities operating as PoT Issuers in a commercial capacity (see Section 1.3): tiered royalty schedule to be published upon patent grant
- End-user implementations that consume but do not issue PoTs: royalty-free

A formal IPR disclosure will be filed on the IETF Datatracker (https://datatracker.ietf.org/ipr/) concurrent with the proposed BoF session. Inquiries may be directed to: peter@kenosian.com

11. References

11.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4", RFC 5905, DOI 10.17487/RFC5905, June 2010, <https://www.rfc-editor.org/info/rfc5905>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8915] Franke, D., Sibold, D., Teichel, K., Dansarie, M., and R. Sundblad, "Network Time Security for the Network

Time Protocol", RFC 8915, DOI 10.17487/RFC8915,
September 2020,
<https://www.rfc-editor.org/info/rfc8915>.

11.2. Informative References

[EIP-712] Ethereum Foundation, "EIP-712: Typed structured data hashing and signing", Ethereum Improvement Proposal, 2017, <https://eips.ethereum.org/EIPS/eip-712>.

[EIGENPHI] EigenPhi Research, "Exclusive data reveals sandwich attacks on Ethereum have waned", CoinTelegraph, December 2025.

[FLASHBOTS] Flashbots, "MEV-Boost and block building on Ethereum", <https://docs.flashbots.net>.

[FLASHBOTS-PROTECT] Flashbots, "Flashbots Protect", <https://protect.flashbots.net>.

[OPENTTT] "OpenTTT: Reference implementation of the TTT SDK", March 2026, <https://github.com/Helm-Protocol/OpenTTT>.

[RFC8179] Arkko, J. and A. Farrel, "Intellectual Property Rights in IETF Technology", BCP 79, RFC 8179, DOI 10.17487/RFC8179, May 2017, <https://www.rfc-editor.org/info/rfc8179>.

[SANDWICH-STUDY] Caradonna, P. et al., "Sandwiched and Silent: Behavioral Adaptation and Private Channel Exploitation in Ethereum MEV", arXiv:2512.17602, December 2024.

[UNISWAP-V4] Adams, H. et al., "Uniswap v4: The Flexibility to Build Anything", Uniswap Blog, January 2025.

Acknowledgments

The authors thank the IETF TSVWG and NTP working groups for their prior work on time synchronization and transport security. The authors also acknowledge the Flashbots research team whose published MEV data provided the empirical foundation for the problem statement in Section 2.

Author’s Address

Heime Jorgen
Kenosian
Email: heime.jorgen@proton.me
URI: https://github.com/Helm-Protocol/OpenTTT