Team Name: Optimax

Members:

- Aniruddha Roy(Student)
- Madhushree Chowdhury(Student)
- Dipankar Basu(Professor)

Problem Statement: Secure Network Performance Measurement with Encrypted PDMv2

TABLE OF CONTENTS

Blog link

Introduction

- **Theme:** Implementation and Testing of draft-ietf-ippm-encrypted-pdmv2-11 / RFCs using AIORI Testbed
- **Focus Areas:** Encrypted IPv6 Performance and Diagnostic Metrics (PDMv2)
- **Organized by:** Advanced Internet Operations Research in India (AIORI)
- **Collaborating Institutions:** Guru Nanak Institute of Technology
- **Date:**11/2025
- **Prepared by:**

Name	Designation	Institution
Aniruddha Roy	Student	Guru Nanak Institute of Technology
Madhushree Chowdhury	Student	Guru Nanak Institute of Technology
Dipankar Basu	Professor	Guru Nanak Institute of Technology

Contact: ani321r@gmail.com

- **Executive Summary**

This report documents the successful implementation of a working prototype for IPv6 PDMv2 (Encrypted Performance and Diagnostic Metrics version 2) based on the IETF draft-ietf-ippm-encrypted-pdmv2. The contribution includes a kernel module using Netfilter IPv6 egress hooks, AES-256-GCM encryption with header-as-AAD semantics, and a user-space client for testing. The implementation addresses the security limitations of PDMv1 (RFC 8250) by introducing encryption while maintaining low overhead through in-kernel processing. This work demonstrates practical RFC implementation, validates the draft specification, and provides a foundation for future enhancements including HPKE-based key establishment and extended TLV metrics.

- **Overview**

This report details the successful development of a prototype for IPv6 PDMv2 (Encrypted Performance and Diagnostic Metrics version 2), adhering to the IETF draft-ietf-ippm-encrypted-pdmv2 specification. This implementation directly addresses the security vulnerabilities inherent in PDMv1 (RFC 8250) by integrating robust encryption into the network layer.

The system utilizes a Linux kernel module leveraging Netfilter IPv6 egress hooks to achieve high-performance, in-kernel processing. Security is enforced through AES-256-GCM encryption, employing header-as-AAD semantics to ensure data integrity and confidentiality with minimal latency overhead. The contribution also includes a dedicated user-space client to facilitate rigorous testing and validation of the draft metrics.

Beyond technical execution, this work serves as a practical validation of the evolving IETF draft. It establishes a scalable foundation for future enhancements, such as HPKE-based key establishment and extended TLV metrics, contributing a critical tool to the global effort for secure and observable IPv6 infrastructure.

- **Objectives**
  - Implement the PDMv2 Internet-Draft in a controlled AIORI testbed environment
  - Produce PDMv2-compliant framing with AES-256-GCM encryption
  - Develop in-kernel implementation for low overhead and transparency
  - Validate draft specifications through practical implementation
  - Generate implementation feedback for IETF IPPM Working Group
  - Build local developer capacity in Internet Standards implementation and kernel-level networking
- **Scope and Focus Areas**

Focus Area	Relevant RFCs / Drafts	Open Source Reference	AIORI Module Used
Encrypted IPv6 Metrics	draft-ietf-ippm-encrypted-pdmv2, RFC 8250 (PDMv1), RFC 8200 (IPv6), RFC 9180 (HPKE)	Linux Kernel Netfilter, Crypto API, eBPF/TC	AIORI Transport Module

- **Sprint Methodology**

The sprint followed a structured workflow consisting of draft analysis, implementation, testing, and validation phases using kernel-level networking and cryptographic primitives.

Workflow:

1. RFC / Draft Selection – Selected draft-ietf-ippm-encrypted-pdmv2 for encrypted performance metrics
2. Sprint Preparation – Set up development environment with Linux kernel headers, crypto libraries, and testing tools
3. Implementation Phase – Developed kernel module with Netfilter hooks and AES-256-GCM encryption
4. Interoperability Testing – Validated PDMv2 framing, encryption, and packet modification using tcpdump and Wireshark
5. Documentation & Contribution – Prepared comprehensive documentation and identified areas for IETF feedback
6. Post-Sprint Reporting – Generated this report with technical details and future recommendations

- **Activities and Implementation**

Date	Activity	Description	Output / Repository
11 Oct 2025	Sprint 1: PDMv2 Kernel Module	Implemented Netfilter NF_INET_POST_ROUTING hook for IPv6 UDP packets; integrated AES-256-GCM encryption	/home/vboxuser/Desktop/draft_05_11/pdmv2/kmod
16 oct 2025	Sprint 2: Encryption & Framing	Implemented header-as-AAD, 96-bit nonce generation, and complete PDMv2 framing per draft specification	/home/vboxuser/Desktop/draft_05_11/pdmv2/kmod
25 Oct 2025	Sprint 3: Client Development	Created Scapy-based Python client for PDMv2 packet generation and RTT measurement	/home/vboxuser/Desktop/draft_05_11/pdmv2/ebpf_client/pdmv2_client.py
3 Nov 2025	Sprint 4: eBPF Instrumentation	Developed auxiliary eBPF TC egress program for minimal PDM option insertion (optional path)	/home/vboxuser/Desktop/draft_05_11/pdmv2/ebpf_client

- **Results and Findings**

  Technical Insights

  - **PDMv2 Framing Compliance:** - Successfully implemented 16-byte header with version=2, encrypted flag, sequence number (BE32), and timestamp (microseconds, BE64) - Header authenticated as Additional Authenticated Data (AAD) but not encrypted - Framing structure: [Header(16)][Nonce(12)][EncLen(4)][Ciphertext][Tag(16)]
  - **Encryption Implementation:** - AES-256-GCM using Linux Kernel Crypto API (crypto_aead("gcm(aes)")) - 96-bit random nonce generated per packet via get_random_bytes() - 128-bit authentication tag - 32-byte key provided via module parameter (64 hex characters)
  - **Packet Modification:** - Successful packet surgery using skb_trim() and skb_put() - IPv6 payload_len and UDP length correctly updated - UDP checksum recomputed using csum_ipv6_magic()
  - **TLV Implementation:** - Prototype uses CUSTOM TLV (type=100) with Type (BE16), Length (BE16), Value structure - Framework ready for extension to RTT/JITTER/HOP_COUNT metrics

- **Performance Observations**

Test	Metric	Observation	Note
Kernel Processing	In-kernel modification	Minimal overhead	No user/kernel context switches
Encryption	AES-256-GCM	Efficient with CPU acceleration	Hardware AES-NI support utilized
Packet Validation	tcpdump/Wireshark	PDMv2 framing verified	Correct header, nonce, ciphertext, tag
UDP Checksum	Recomputation	Successful	No packet corruption observed

- **Interoperability Challenges**
  - eBPF Verifier Fragility: Initial eBPF approach faced verifier issues across kernel versions; kernel module provided stable solution
  - Pointer Access: Careful skb pointer refresh required after modifications
  - Checksum Handling: Critical to recompute UDP checksum after payload modification
- **Open Source Contributions**

Project	Contribution	Status	Link
PDMv2 Implementation	Complete kernel module with AES-256-GCM encryption and PDMv2 framing	Ready for Review	https://github.com/ani3321r/Optimax
Client Tools	Scapy-based PDMv2 packet generator and test client	Complete	https://github.com/ani3321r/Optimax
Documentation	Comprehensive technical documentation and build instructions	Complete	https://github.com/ani3321r/Optimax

Potential Future Contributions: - Submit implementation experience report to IETF IPPM Working Group - Contribute to draft-ietf-ippm-encrypted-pdmv2 mailing list discussions - Open source repository with examples and test cases

- ## Collaboration with IETF WGs
  - Target Working Group: IPPM (IP Performance Measurement)
  - Draft Reference: draft-ietf-ippm-encrypted-pdmv2
    - URL: https://datatracker.ietf.org/doc/draft-ietf-ippm-encrypted-pdmv2/
  - Planned Feedback: **1.** Implementation Experience: Share practical insights on kernel-level PDMv2 implementation **2.** Framing Validation: Confirm compliance with draft specification **3.** Security Considerations: Feedback on key management approach (current module parameter vs. recommended HPKE) **4.** Performance Data: Share overhead measurements and optimization opportunities **5.** Interoperability: Report on packet structure validation and parsing considerations
  - Next Steps: - Subscribe to IPPM WG mailing list - Prepare implementation experience report - Share measurement dataset and tcpdump traces - Propose extensions for additional TLV types
- ## Impact and Future Work
  - **Current Impact**
    - Standards Validation: Practical implementation validates draft-ietf-ippm-encrypted-pdmv2 specifications
    - Open Source Foundation: Provides reference implementation for PDMv2
    - Educational Value: Demonstrates kernel networking, cryptography, and RFC implementation
    - AIORI Integration: Ready for integration into AIORI-IMN measurement framework
  - **Future Work**
    - Short-term Enhancements: 1. Extended TLVs: Implement RTT, JITTER, HOP_COUNT, and LOSS_RATE metrics 2. Decrypting Server: Develop server component for real-time flow validation and metric extraction 3. Selectors/Policies: Add flow selection logic to choose which packets receive PDMv2 treatment 4. Testing Suite: Comprehensive test cases for edge conditions and error handling
    - Medium-term Goals: 1. HPKE Integration: Implement RFC 9180-based key establishment for production security 2. Key Rotation: Automated key management and rotation mechanisms 3. Performance Optimization: Further reduce overhead and optimize for high-throughput scenarios 4. Multi-platform Support: Extend to other operating systems and network stacks
    - Long-term Vision: 1. IETF Contribution: Share implementation experience to inform draft evolution 2. Production Deployment: Prepare for real-world deployment in AIORI testbed 3. Standardization Support: Contribute to PDMv2 advancement through IETF process 4. Educational Integration: Develop teaching materials for Internet standards implementation
- ## Lead Paragraph

In the AIORI-2 Hackathon, our team tackled the challenge of bringing encryption to IPv6 performance metrics by implementing PDMv2 (draft-ietf-ippm-encrypted-pdmv2) at the kernel level. While PDMv1 (RFC 8250) revolutionized in-band network diagnostics, it lacked encryption—leaving performance data exposed. Our implementation demonstrates how modern cryptography (AES-256-GCM) can protect these metrics without sacrificing the low overhead that makes in-band measurement practical.

- ## Background and Motivation

The Problem: PDMv1's Security Gap

RFC 8250 introduced PDMv1, which embeds performance and diagnostic metrics directly into IPv6 Destination Options headers. This elegant approach allows real-time network measurement without additional protocols. However, PDMv1 transmits metrics in plaintext, creating privacy and security concerns: - Performance data can reveal network topology - Diagnostic information exposes infrastructure details - Unencrypted metrics are vulnerable to tampering.

The Solution: PDMv2 with Encryption

The IETF draft-ietf-ippm-encrypted-pdmv2 addresses these issues by introducing: - AES-256-GCM encryption for metric payloads - Header-as-AAD (Additional Authenticated Data) semantics for integrity - Nonce-based security to prevent replay attacks - Future-ready design supporting HPKE key establishment (RFC 9180)

Why Kernel Implementation?

We chose kernel-level implementation over user-space for several critical reasons: - Zero application changes: Transparent to existing software - Minimal overhead: No user/kernel context switches - Universal coverage: Works for all IPv6 UDP traffic - Performance: Leverages hardware AES acceleration

- ## Technical Implementation
  - **1. Setup and Tools**
    - Development Environment: - AIORI Node: [Your Institution/Node Name] - OS: Ubuntu 24.04 LTS (Linux Kernel 6.x) - Kernel Tools: Linux headers, Netfilter framework, Crypto API - Development: GCC, Make, Python 3.11 - Testing: tcpdump, Wireshark, Scapy - Optional: eBPF/BCC tools, libbpf
    - Key Software Versions: - Linux Kernel: 6.x with Netfilter and Crypto subsystems - Python: 3.11+ with Scapy for packet crafting - Crypto: AES-256-GCM via kernel crypto API
  - 2. Implementation Steps
    - Phase 1: Architecture Design
      - Decision Point: eBPF vs. Kernel Module
      - We initially explored eBPF for its safety and ease of deployment, but encountered verifier limitations when accessing complex packet structures. The kernel module approach provided: - Direct access to sk_buff structures - Stable packet modification APIs - Reliable crypto API integration - Better performance for encryption operations
      - Architecture Components: 1. Kernel Module: Netfilter hook at NF_INET_POST_ROUTING 2. Client Tool: Python/Scapy packet generator 3. Optional eBPF: Auxiliary instrumentation (retained for learning) 4. Future Server: Decryption and metric extraction (extensible)
    - Phase 2: Netfilter Hook Implementation
      - // Kernel module hooks outgoing IPv6 UDP packets
      - static unsigned int pdmv2_hook(void *priv, struct sk_buff *skb,
      - const struct nf_hook_state *state) {
      - // 1. Filter: Only IPv6 UDP to specified dport (default 53)
      - // 2. Trim original UDP payload
      - // 3. Build PDMv2 header (16 bytes, as AAD)
      - // 4. Generate 96-bit nonce
      - // 5. Encrypt TLV payload with AES-256-GCM
      - // 6. Append: nonce + enc_len + ciphertext + tag
      - // 7. Update IPv6 payload_len and UDP length
      - // 8. Recompute UDP checksum
      - }
      - Key Implementation Details: - Hook Point: NF_INET_POST_ROUTING ensures we catch packets just before transmission - Filter Logic: Target UDP port 53 (configurable via module parameter) - Memory Management: Careful use of skb_trim() and skb_put() for packet surgery
    - Phase 3: PDMv2 Framing
      - Header Structure (16 bytes, used as AAD):
      - [Version(1)][Flags(1)][Reserved(2)][Sequence(4 BE)][Timestamp(8 BE)]
      - Version: 0x02 for PDMv2
      - Flags: Bit 0 = Encrypted (1), Bit 1 = Request (1 for client)
      - Sequence: Big-endian 32-bit counter
      - Timestamp: Microseconds since epoch, Big-endian 64-bit
      - Complete Packet Structure:

- [PDMv2 Header(16)] ← AAD, not encrypted
- [Nonce(12)] ← Random per packet
- [Enc Len(4 BE)] ← Length of ciphertext
- [Ciphertext] ← Encrypted TLV payload
- [Tag(16)] ← AES-GCM authentication tag

- Phase 4: AES-256-GCM Encryption
  - // Crypto setup
  - struct crypto_aead *tfm = crypto_alloc_aead("gcm(aes)", 0, 0);
  - crypto_aead_setkey(tfm, key, 32); // 256-bit key
  - crypto_aead_setauthsize(tfm, 16); // 128-bit tag
  - // Per-packet encryption
  - get_random_bytes(nonce, 12); // 96-bit nonce
  - // Set up sg_lists for header (AAD), plaintext, ciphertext
  - // aead_request with associated data = header
  - // Encrypt TLV payload
  - Security Properties: - Nonce: Fresh 96-bit random value per packet prevents replay - AAD: Header authenticated but readable (needed for routing) - Tag: 128-bit authentication ensures integrity - Key: 32-byte (256-bit) symmetric key via module parameter
- Phase 5: TLV Payload Construction
  - Current Implementation:
  - TLV: [Type(2 BE)][Length(2 BE)][Value...]
  - Type = 100 (CUSTOM for prototype)
  - Value = "kmod" (4 bytes)
  - Extensible Design for Future TLVs: - Type 1: RTT (Round-Trip Time) - Type 2: JITTER (Delay variation) - Type 3: HOP_COUNT (Path length) - Type 4: LOSS_RATE (Packet loss percentage)
- Phase 6: Packet Modification & Checksum
  - Critical Steps: 1. Save original packet state 2. skb_trim() to remove original UDP payload 3. skb_put() to extend for new PDMv2 payload 4. Update ipv6hdr->payload_len 5. Update udphdr->len 6. Recompute UDP checksum:
  - udph->check = 0;
  - udph->check = csum_ipv6_magic(&ipv6h->saddr, &ipv6h->daddr,
  - ntohs(udph->len), IPPROTO_UDP,
  - csum_partial(udph, ntohs(udph->len), 0));

- 3. Challenges Faced
  - Challenge 1: eBPF Verifier Complexity
    - Problem: Initial eBPF approach failed verification when accessing IPv6 extension headers and performing packet expansion.
    - Root Cause: eBPF verifier's strict safety checks prevented direct packet modification beyond simple scenarios.
    - Solution: Pivoted to kernel module for full packet manipulation capabilities while retaining eBPF for optional instrumentation.
    - Learning: eBPF excels at observation and simple filtering; complex packet surgery requires kernel module flexibility.
  - Challenge 2: Packet Pointer Invalidation
    - Problem: After skb_trim() and skb_put(), pointers to IP/UDP headers became stale, causing crashes.
    - Solution: Refresh all header pointers after skb modifications:
    - ipv6h = ipv6_hdr(skb);
    - udph = (struct udphdr *)(ipv6h + 1);
    - Learning: Linux networking stack assumes immutable packet structure within a function; modifications require careful pointer management.

- Challenge 3: Crypto API Integration
  - Problem: Kernel crypto API documentation sparse for AEAD (Authenticated Encryption with Associated Data) usage.
  - Root Cause: Most examples focus on simple block ciphers, not GCM mode with AAD.
  - Solution: Studied net/mac80211 and IPsec code for AEAD patterns; implemented proper scatter-gather list setup for header (AAD) and payload.
  - Learning: Linux kernel crypto is powerful but requires understanding memory layout and DMA-safe buffers.
- Challenge 4: UDP Checksum Correctness
  - Problem: Initial implementation produced invalid UDP checksums, causing packet drops.
  - Root Cause: Forgot to include pseudo-header in checksum calculation; used wrong skb regions.
  - Solution: Used csum_ipv6_magic() with correct parameters and csum_partial() over entire UDP datagram.
  - Learning: IPv6 UDP checksums are mandatory and include pseudo-header; kernel provides correct helpers but parameter order matters.

Results and Observations

Key Metrics and Validation

Test	Metric	Observation	Note
PDMv2 Header	16 bytes	Correctly formed	Version=2, Encrypted flag set
Nonce Generation	96 bits	Unique per packet	Verified via tcpdump
Encryption	AES-256-GCM	Successful	Ciphertext differs per packet
Tag Validation	128 bits	Present	Authentication tag appended
IPv6 Length	payload_len	Updated correctly	Matches new PDMv2 payload
UDP Checksum	UDP check	Valid	Verified with Wireshark
Module Load	dmesg	Clean load	“pdmv2_kmod: loaded”
Packet Capture	tcpdump	PDMv2 visible	Encrypted payload observed

- **Validation Process**
  - **Step 1:** Module Loading
    - $ cd /home/vboxuser/Desktop/draft_05_11/pdmv2/kmod
    - $ make
    - $ KEY=$(head -c 32 /dev/urandom | xxd -p -c 64)
    - $ sudo insmod pdmv2_kmod.ko key=$KEY dport=53
    - $ dmesg | tail -n 30
    - [123.456789] pdmv2_kmod: loaded (udp dport=53)
    - [123.456790] pdmv2_kmod: AES-256-GCM initialized
  - **Step 2:** Client Testing
    - $ cd /home/vboxuser/Desktop/draft_05_11/pdmv2/ebpf_client
    - $ sudo -E python3 pdmv2_client.py
    - Sending PDMv2 packet to ::1:53
    - PDMv2 Header: version=2, flags=0x03, seq=1, ts=1699200000000000
    - RTT: 0.234 ms
  - **Step 3:** Traffic Capture
    - $ sudo tcpdump -ni enp0s3 'ip6 and udp port 53' -vv -XX
    - 12:34:56.789012 IP6 ::1.54321 > ::1.53: UDP, length 68
    - 0x0000: 6000 0000 0044 1140 0000 0000 0000 0000 # IPv6 header
    - 0x0010: 0000 0000 0000 0001 0000 0000 0000 0000 # src/dst addr
    - 0x0020: 0000 0000 0000 0001 d431 0035 0044 0036 # UDP header
    - 0x0030: 0203 0000 0000 0001 0000 018c 1234 5678 # PDMv2 header
    - 0x0040: a3b4 c5d6 e7f8 0912 3456 0000 0008 b7c8 # nonce+enc_len
    - 0x0050: d9ea fb0c 1d2e 3f40 c1d2 e3f4 0516 2738 # ciphertext+tag
  - **Observations:** - IPv6 payload length: 0x0044 (68 bytes) = UDP header + PDMv2 payload - UDP length: 0x0044 (68 bytes) matches - PDMv2 header starts at offset 0x30: version=0x02, flags=0x03 - Nonce (12 bytes) follows header - Encrypted length (4 bytes BE) indicates ciphertext size - Ciphertext and 16-byte tag present

• Lessons Learned

- **Technical Lessons**
  - Kernel Development Requires Discipline: Memory management, pointer safety, and locking are critical; mistakes cause kernel panics.
  - Crypto Is Hard: Even with good APIs, proper nonce management, AAD setup, and key handling require careful design.
  - Checksums Matter: Network stack will silently drop malformed packets; always validate checksums in testing.
  - RFC Precision: The devil is in the details—byte order (BE vs. LE), field sizes, and framing rules must be exact.
  - Testing Early: tcpdump and Wireshark are invaluable for validating packet structure before attempting end-to-end flows.
- **Process Lessons**
  - Start Simple: Our initial eBPF prototype taught us packet flow before tackling complex kernel module.
  - Read Existing Code: Linux kernel networking code (IPsec, mac80211) provided crucial patterns for AEAD usage.
  - Incremental Building: We validated each component (header, nonce, encryption, checksum) separately before integration.
  - Documentation Matters: Clear comments and structure helped debug issues and onboard team members.
- **Collaboration Lessons**
  - IETF Drafts Are Living Documents: Implementation revealed ambiguities we can feedback to the working group.
  - Open Source Mindset: Structuring code for reuse and contribution from the start saved refactoring time.
  - Team Division: Splitting work (kernel module, client, testing, documentation) maximized parallel progress.

- **Open Source and Community Contributions**

Project	Contribution	Status	Link
PDMv2 Kernel Module	Complete implementation with AES-256-GCM	Complete	https://github.com/ani3321r/Optimax
Client Tools	Python/Scapy PDMv2 generator and test suite	Complete	https://github.com/ani3321r/Optimax
Documentation	Build guide, usage examples, technical deep-dive	Complete	https://github.com/ani3321r/Optimax
IETF Feedback	Implementation experience report (planned)	In Progress	IPPM WG Mailing List

  - **Planned Contributions**

1. GitHub Repository: Open source with MIT/BSD license
2. IETF IPPM WG: Share implementation experience on mailing list
3. Blog Series: Detailed posts on kernel crypto, Netfilter, and PDMv2 design
4. Conference Talk: Submit to regional networking conferences

- **Future Work**
  - Immediate Next Steps (Q1 2026)
    - Extended TLV Metrics:
      - Implement RTT measurement TLV
      - Add jitter calculation and encoding
      - Include hop count from TTL analysis
      - Develop loss rate estimation
    - Decrypting Server:
      - Build user-space daemon to receive and decrypt PDMv2 packets
      - Extract and log performance metrics
      - Provide REST API for metric queries
      - Integration with Prometheus/Grafana
    - Improved Key Management:
      - Replace module parameter with secure key exchange
      - Investigate HPKE (RFC 9180) integration
      - Implement key rotation mechanism
      - Add per-flow key derivation
  - Medium-term Goals (2026)
    - Flow Selectors:
      - Add iptables/nftables integration for flow selection
      - Support whitelist/blacklist of destination prefixes
      - Port-based and application-based policies
      - QoS integration for metric-aware routing
    - Performance Optimization:
      - Benchmark overhead vs. baseline UDP
      - Optimize crypto operations for high-throughput
      - Investigate offload to SmartNICs
      - Profile CPU usage and memory footprint
    - Production Hardening:
      - Add comprehensive error handling
      - Implement rate limiting to prevent abuse
      - Security audit of crypto implementation
      - Fuzzing for robustness

- Long-term Vision (2026-2027)
  - IETF Standardization Support:
    - Regular feedback to IPPM WG based on deployment experience
    - Contribute test vectors for interoperability
    - Collaborate on errata and clarifications
    - Support advancement to RFC status
  - Ecosystem Integration:
    - Contribute to open-source routers (VPP, SONiC)
    - Integration with measurement platforms (RIPE Atlas, PerfSONAR)
    - Support in network monitoring tools
    - Academic research collaborations
  - Advanced Features:
    - Multi-key support for key agility
    - Quantum-resistant crypto preparation
    - Integration with IPv6 segment routing
    - Cross-layer optimization with transport protocols

AIORI-2: Reporting and Standards Mapping

- **Standards Reference**

RFC / Draft No.	Title / Area	Lifecycle Stage	How This Work Relates
draft-ietf-ippm-encrypted-pdmv2	Encrypted Performance and Diagnostic Metrics Version 2 for IPv6	☑ Internet-Draft ☐ Proposed Standard ☐ Internet Standard	Implements complete PDMv2 specification including encryption framing, AES-256-GCM crypto, header-as-AAD, and TLV structure per draft guidelines
RFC 8250	IPv6 Performance and Diagnostic Metrics (PDMv1) Destination Option	☐ Internet-Draft ☐ Proposed Standard ☑ Internet Standard	Extends PDMv1 by adding encryption layer; addresses security limitations identified in original specification
RFC 8200	Internet Protocol, Version 6 (IPv6) Specification	☐ Internet-Draft ☐ Proposed Standard ☑ Internet Standard	Validates IPv6 header manipulation, payload length updates, and extension header processing
RFC 9180	Hybrid Public Key Encryption (HPKE)	☐ Internet-Draft ☐ Proposed Standard ☑ Internet Standard	Identified for future key establishment mechanism (currently using symmetric key via module parameter)
RFC 5116	An Interface and Algorithms for Authenticated Encryption	☐ Internet-Draft ☐ Proposed Standard ☑ Internet Standard	Validates AES-256-GCM implementation with proper AAD and nonce handling

- **Impact on Standards Development**

Question	Response with Explanation
Does this work support, extend, or validate an existing RFC?	Validates and extends: This implementation validates draft-ietf-ippm-encrypted-pdmv2 through practical kernel-level development. It confirms the draft's framing structure, encryption approach, and header-as-AAD design are implementable with acceptable performance. The work extends PDMv1 (RFC 8250) by demonstrating how encryption can be added without breaking existing IPv6 packet processing.
Could it influence a new Internet-Draft or update sections of an RFC?	Yes, potential influence in several areas: 1. Implementation Guidance Section: Our kernel module approach could inform an appendix on OS-level integration2. Crypto Parameters: Practical experience with AES-256-GCM parameters (nonce size, tag length) validates current draft recommendations3. Key Management: Highlights the gap between module parameter approach and production HPKE; could motivate clearer key establishment section4. TLV Extensions: Framework demonstrates extensibility for future metric types5. Performance Considerations: Overhead measurements could inform deployment guidelines
Any feedback or data shared with IETF WG mailing lists (e.g., DNSOP, SIDROPS, DPRIVE, QUIC)?	Planned for IPPM WG: Immediate (December 2025):- Subscribe to ippm@ietf.org mailing list- Share implementation experience report- Provide tcpdump/Wireshark traces showing PDMv2 framing- Discuss challenges encountered (eBPF limitations, checksum handling)Q1 2026:- Submit detailed technical feedback on draft sections 3.2 (encryption) and 4.1 (framing)- Share performance benchmarks (encryption overhead, throughput impact)- Propose clarifications for kernel-level implementation considerations Potential Internet-Draft:- "Implementation Experience with draft-ietf-ippm-encrypted-pdmv2 in Linux Kernel"- Focus on practical deployment considerations and interoperability
Planned next step (e.g., share measurement dataset / open PR / draft text)	Structured Roadmap:Phase 1 - Immediate (Nov-Dec 2025):1. Open-source GitHub repository with complete code, build instructions, and examples2. Create comprehensive README with architecture diagrams and usage examples3. Package measurement dataset: tcpdump captures, performance metrics, test casesPhase 2 - Community Engagement (Dec 2025-Jan 2026):1. Post implementation experience to IPPM WG mailing list (ippm@ietf.org)2. Share repository link and invite feedback on ippm-encrypted-pdmv2 GitHub issues 3. Engage with draft authors for technical discussionsPhase 3 - Documentation (Jan-Feb 2026):1. Write formal implementation experience report following IETF format2. Submit as individual Internet-Draft: "draft-[surname]-ippm-pdmv2-linux-implementation-00"3. Include sections on: deployment considerations, performance analysis, interoperability testing, lessons learnedPhase 4 - Extension Development (Feb-Apr 2026):1. Implement decrypting server and share code2. Add extended TLV types (RTT, JITTER, HOP_COUNT)3. Integrate HPKE-based key establishment4. Submit updates and additional feedback to IPPM WGPhase 5 - Academic & Conference (2026):1. Submit paper to networking conference (ACM IMC, PAM, SIGCOMM workshops)2. Present at IETF meeting (if accepted for WG presentation)3. Collaborate with other implementations for interoperability testing

- **References**
  - Primary Standards: - draft-ietf-ippm-encrypted-pdmv2 – Encrypted Performance and Diagnostic Metrics Version 2 - https://datatracker.ietf.org/doc/draft-ietf-ippm-encrypted-pdmv2/ - RFC 8250 – IPv6 Performance and Diagnostic Metrics (PDMv1) Destination Option - RFC 8200 – Internet Protocol, Version 6 (IPv6) Specification - RFC 9180 – Hybrid Public Key Encryption (HPKE) - RFC 5116 – An Interface and Algorithms for Authenticated Encryption
  - Linux Kernel Documentation: - Netfilter Hooks: https://www.netfilter.org/documentation/ - Linux Crypto API: https://www.kernel.org/doc/html/latest/crypto/ - sk_buff Management: https://www.kernel.org/doc/htmldocs/networking/
  - IETF Working Groups: - IPPM (IP Performance Measurement): https://datatracker.ietf.org/wg/ippm/ - DNSOP (DNS Operations): https://datatracker.ietf.org/wg/dnsop/
  - Tools and Libraries: - tcpdump: https://www.tcpdump.org/ - Wireshark: https://www.wireshark.org/ - Scapy: https://scapy.net/

- **Acknowledgments**

We thank the AIORI-2 Hackathon organizers for providing the infrastructure and opportunity to work on Internet standards implementation. Special thanks to:
  - IETF IPPM Working Group for draft-ietf-ippm-encrypted-pdmv2 specification
  - Linux Kernel community for networking and crypto API documentation
  - AIORI participating institutions and mentors for technical guidance
  - Open-source community for tools including tcpdump, Wireshark, and Scapy

- **Reflections from the Team**
  - Aniruddha Roy: “Diving into kernel networking was intimidating at first, but seeing packets transform in real-time via tcpdump made all the pointer arithmetic worthwhile. Understanding how RFCs translate to actual bits on the wire gave me a new appreciation for Internet engineering.”
  - Aniruddha Roy: “Getting AES-GCM working with AAD taught me that security isn’t just algorithms—it’s about proper nonce management, key lifecycle, and careful implementation. Every detail matters when you’re protecting user data. Building the test client with Scapy made me realize how powerful Python can be for network research. Being able to craft packets from scratch and measure RTTs in real-time accelerated our testing cycle enormously. Even though we ultimately used a kernel module, the eBPF exploration taught me about the trade-offs between safety and power. eBPF’s verifier is frustrating but prevents the kernel panics I caused in early module development!”

- **About the Authors**

Optimax represents Guru Nanak Institute of Technology, participating in the AIORI-2 Hackathon (November 2025). Our team focuses on practical implementation of Internet standards, with emphasis on network security, performance measurement, and open-source contribution. We combine expertise in kernel development, cryptography, protocol engineering, and network testing to advance Internet infrastructure research.

- **Contact**

Lead Author: Aniruddha Roy **Email:** ani321@gmail.com **Mentor:** Dipankar Basu