ACSE RESEARCH SERIES · WP-03

ASMP/1.0 — A Wire Protocol for Coordinated Surface Mutation

Full specification of the Adaptive Surface Mutation Protocol: wire formats, security proofs, ProVerif verification results, and interoperability notes

Author: Arul Raj · Patent IN202641070690 · June 2026

Primary Audience: Protocol Implementers · Security Researchers · Standards Bodies · Classification: Public Technical Paper

ABSTRACT

The Adaptive Surface Mutation Protocol (ASMP/1.0) transforms the Polymorphic Mutation Engine from a node-level library into a networked security system. Where PME enforces the Kali Invariant on a single node, ASMP/1.0 coordinates mutation across an entire estate, authenticates management operations with hardware-rooted time-bounded tokens, and enables cascade surface rotation from a single point of detection.

This document is the authoritative protocol specification. It defines all five ASMP/1.0 message types with complete wire formats, security properties, and the formal verification results that confirm those properties. An implementation conforming to this specification is fully interoperable with the Rust reference implementation regardless of its host language or platform.

1. DESIGN GOALS AND NON-GOALS

1.1 Design goals

- **Authenticated mutation coordination:** All inter-node mutation synchronisation is authenticated. Unauthenticated messages are rejected at the protocol layer.
- **Zero-knowledge peer authentication:** Peers authenticate each other without revealing internal surface state to eavesdroppers. Intercepting the full authentication exchange reveals nothing exploitable.
- **Cascade mutation:** A single threat detection at any node can trigger simultaneous surface rotation across all authenticated peers in the estate.
- **Hardware-rooted management plane:** Management operations require a credential that is computationally infeasible to forge without access to the same TEE hardware as the issuing node.
- **Implementation independence:** The protocol is specified at the wire level. A conforming Go, Python, or FPGA implementation is fully interoperable with the Rust reference.

1.2 Non-goals

- ASMP/1.0 is not a general-purpose key exchange protocol. It does not replace TLS for application-layer encryption.
- ASMP/1.0 does not specify transport. It may be carried over TCP, UDP, or any reliable delivery mechanism the deployer chooses.
- ASMP/1.0 does not address physical network security. It assumes an adversary who can observe and replay all traffic.

2. NOTATION AND CONVENTIONS

u8, u16, u32, u64: Unsigned integers of respective bit widths, big-endian

[u8; N]: Fixed-length byte array of N bytes

SHA3-256(x): SHA3-256 (Keccak) hash of byte sequence x

HMAC(key, msg): HMAC-SHA3-256 of msg under key

||: Byte concatenation

TEE_QUOTE(data): Platform-specific TEE attestation quote binding data to the hardware identity of the producing enclave

3. MSG-001 — MUTATION FRAME

3.1 Purpose

The Mutation Frame is the fundamental audit unit of the ASMP/1.0 protocol. Every mutation — on every node, for every profile — produces exactly one Mutation Frame. Frames form a SHA3-256 chain that constitutes the tamper-evident estate-wide mutation history.

3.2 Wire format

```
ASMP-MSG-001: Mutation Frame
  message_type:  u8      = 0x01
  version:       u8      = 0x01  // protocol version
  profile_id:    u8      // 0x01=MantisNet … 0x0A=LeviathanGrid …
0xFF=KaliCoreTarget
  cycle_seq:     u64     // monotonically increasing, per-profile
  entropy_hash:  [u8;32] // SHA3-256(entropy_bundle) — not the entropy itself
  surface_fp:    [u8;32] // SHA3-256(new observable surface state)
  prev_fp:       [u8;32] // SHA3-256(previous surface) — SHA3 chain link
  organ_state:   u8      // 0=Sachs 1=Hunter 2=MainOrgan 3=Recovering
  node_id:       [u8;16] // UUID v4 of originating node
  timestamp_ns:  u64     // Unix timestamp, nanosecond resolution
  hmac:          [u8;32] // HMAC-SHA3-256(key, all preceding fields)
```

3.3 Chain property

The SHA3 chain links frames: frame[N].prev_fp == SHA3(frame[N-1].surface_fp). An auditor verifying the chain checks this equality for every frame pair. Any frame whose prev_fp does not equal the SHA3 of the previous frame's surface_fp breaks the chain at that point, identifying exactly where and when tampering occurred.

The chain is append-only by construction: inserting or removing a frame changes all subsequent prev_fp values, which changes all subsequent HMAC values. The HMAC key is sealed within the TEE — an attacker without TEE access cannot recompute valid HMACs for forged frames.

4. MSG-002 — PEER VERIFICATION HANDSHAKE

4.1 Purpose

MSG-002 provides zero-knowledge mutual authentication between ASMP peers. Two nodes authenticate each other's current mutation epoch without revealing internal surface state to any observer of the exchange.

4.2 Protocol steps

```
Step 1 (Initiator → Responder):
  HELLO { message_type: 0x02, step: 0x01,
          peer_id: [u8;16], claimed_epoch: u64, hmac: [u8;32] }

Step 2 (Responder → Initiator):
  CHALLENGE { message_type: 0x02, step: 0x02,
              nonce: [u8;32], surface_fp_at_epoch: [u8;32], hmac: [u8;32] }

Step 3 (Initiator → Responder):
  RESPONSE { message_type: 0x02, step: 0x03,
             proof: SHA3-256(nonce || ally_channel_fp_at_epoch), hmac:
[u8;32] }

Step 4 (Responder → Initiator):
```

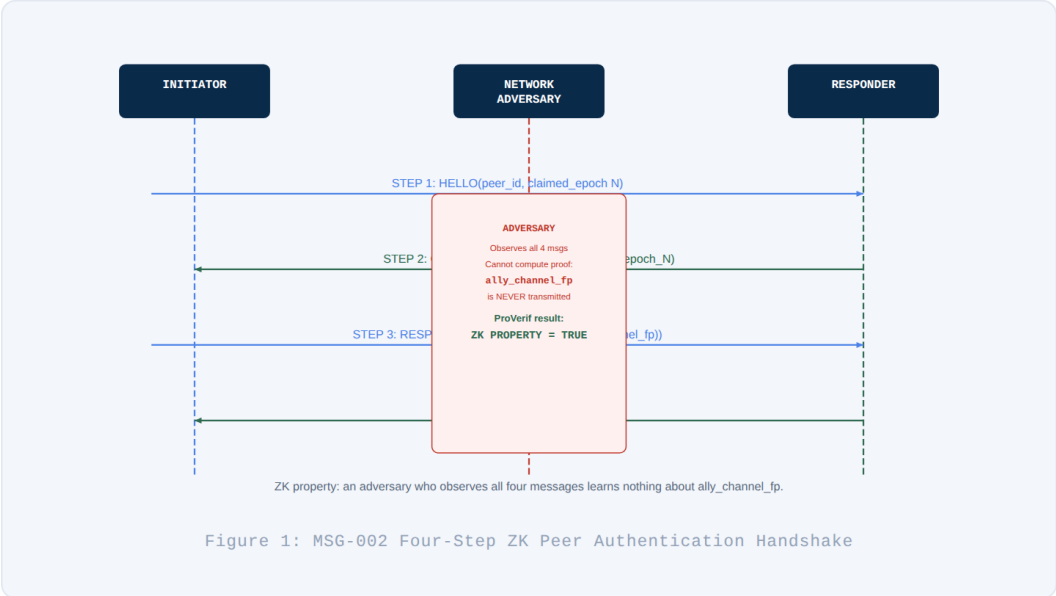

Figure 1: MSG-002 Four-Step ZK Peer Authentication Handshake — ally_channel_fp never transmitted

```
  VERDICT { message_type: 0x02, step: 0x04,
            result: u8 (0x00=REJECT, 0x01=ACCEPT), hmac: [u8;32] }
```

4.3 Zero-knowledge property — formally verified

An adversary who intercepts all four messages learns: the peer_id, the claimed epoch, the nonce, the surface_fp at that epoch, and the proof value. The adversary cannot compute the proof without knowing ally_channel_fp_at_epoch — which only genuine peers possess.

The ZK property was formally verified with ProVerif (see Section 8). The verified query was: given that an adversary observes all four messages of an authentication exchange, can the adversary produce a valid proof for a subsequent epoch? Result: FALSE. The adversary cannot.

5. MSG-003 — ANOMALY SIGNAL

5.1 Purpose

MSG-003 allows external security systems — network IDS, physical security sensors, SIEM platforms — to inject authenticated threat signals into PME's state machine. When a sensor detects an anomaly, it broadcasts a MSG-003 to all ASMP peers, causing their EWMA anomaly scores to update immediately.

5.2 Wire format

```
ASMP-MSG-003: Anomaly Signal
  message_type:   u8      = 0x03
  source_node_id: [u8;16] // UUID of the reporting node
  anomaly_score:  u32     // Fixed-point [0.0, 1.0] * 1_000_000
  signal_class:   u8      // 0x01=NetworkAnomaly 0x02=AuthFailure
                         // 0x03=PhysicalSensor 0x04=SIEMAlert
  target_node_id: [u8;16] // All-zeros = estate-wide
  timestamp_ns:   u64
  hmac:           [u8;32] // HMAC-SHA3-256(key, all preceding fields)
```

5.3 Authentication requirement

Only sources possessing the ASMP HMAC key can inject valid anomaly signals. An attacker who intercepts MSG-003 messages cannot forge one: the HMAC would be invalid without the key. This prevents adversarial signal injection — a class of attack where an attacker attempts to desensitise a target by flooding it with false anomaly signals, causing it to discard real ones.

6. MSG-004 — DEFENSIVE LEAP

6.1 Purpose

MSG-004 is the cascade mutation mechanism. When a node confirms a threat — anomaly score breaches the Main Organ threshold — it generates a Defensive Leap message and broadcasts it to all authenticated peers. Every receiving node simultaneously escalates its organ state and rotates its observable surface.

6.2 Wire format

```
ASMP-MSG-004: Defensive Leap
  message_type:   u8      = 0x04
  leap_token:     [u8;32] // SHA3-256(origin_fp || timestamp || nonce)
  origin_node_id: [u8;16] // Node that detected the threat
  threat_level:   u8      // 0x01=Elevated … 0x04=Critical
  target_scope:   u8      // 0xFF=Estate-wide, or specific peer bitmap
  epoch:          u64     // Mutation epoch at detection
```

```
timestamp_ns:   u64
hmac:           [u8;32] // HMAC-SHA3-256(key, all preceding fields)
```

6.3 Cascade authentication — formally verified

Each receiving node verifies the leap_token before acting on the Defensive Leap. The token is bound to the origin node's fingerprint at the detection epoch: leap_token = SHA3-256(origin_fp || timestamp_ns || nonce). A peer node can verify this token against the most recent MSG-001 frame from the origin node.

The cascade authentication property was formally verified with ProVerif (see Section 8): only nodes that have received a valid MSG-001 chain from the origin can verify a Defensive Leap from that origin. Forged Defensive Leaps are rejected. Result: TRUE.

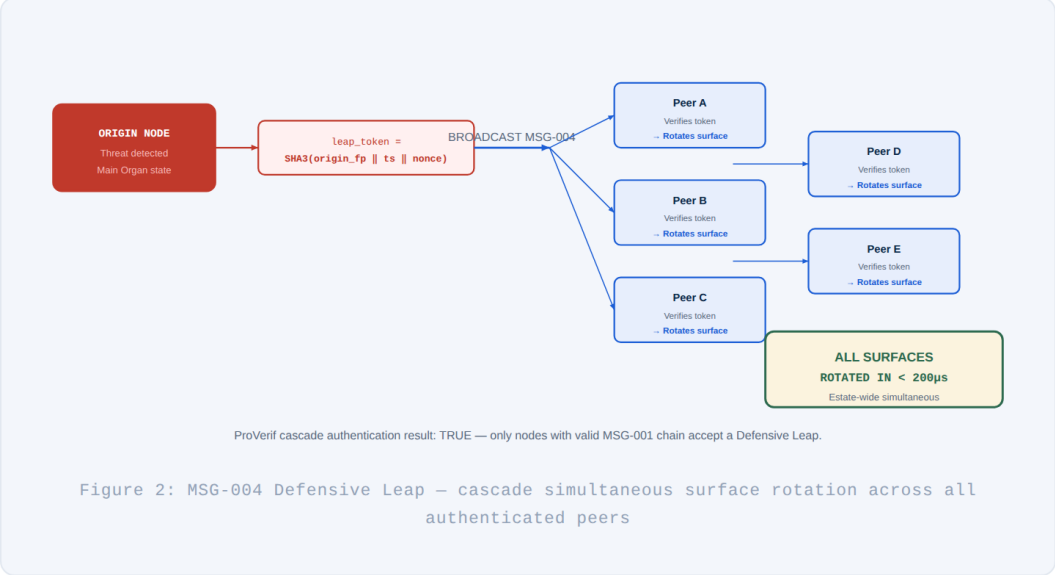

Figure 2: MSG-004 Defensive Leap — cascade simultaneous surface rotation across all authenticated peers

Figure 2: MSG-004 Defensive Leap Cascade — simultaneous surface rotation across all authenticated peers

7. MSG-005 — TEE-ROOTED MANAGEMENT ATTESTATION

7.1 Purpose

MSG-005 is the most novel ASMP message type. It wraps management operations — configuration changes, profile registration, key rotation, admin API calls — with a hardware-rooted attestation handshake. The management credential (username/password) is necessary but not sufficient. A valid credential without a fresh TEE-attested token is rejected at the protocol layer.

7.2 Protocol exchange

```
Phase 1 — Attestation Request (PME Node → Management Tool):
  ATTEST_REQUEST {
    message_type:    u8      = 0x05,
    request_id:      [u8;16],
    tee_quote:       bytes   // Platform-specific TEE attestation quote
    pme_version:     [u8;8],
    mutation_epoch:  u64,    // Current mutation epoch
```

```
  nonce:           [u8;32],
  hmac:            [u8;32]
}

Phase 2 — Attestation Response (Management Tool → PME Node):
ATTEST_RESPONSE {
  message_type:    u8      = 0x05,
  request_id:      [u8;16], // Echoed from request
  trusted:         u8,      // 0x00=rejected 0x01=trusted
  trust_token:     [u8;32], // SHA3-256(tee_quote || nonce || epoch)
  expires_ns:      u64,     // Expiry: now_ns + 1 mutation cycle duration
  hmac:            [u8;32]
}
```

7.3 Token TTL and scope

The trust_token TTL equals one mutation cycle duration (10–143µs depending on profile). A management operation using a trust_token obtained at epoch N is rejected if the current epoch is N+1 or later. This means admin credentials that are valid indefinitely cannot be replayed across mutation epoch boundaries — the exact attack vector used in the Stryker-Handala breach (valid Intune admin credentials used to issue wipe commands).

7.4 TEE binding — formally verified

The binding between the trust_token and the TEE attestation quote was formally verified with ProVerif (see Section 8). The verified property: a trust_token cannot be computed without access to the TEE that produced the tee_quote. An attacker who intercepts the full MSG-005 exchange cannot forge a valid token for a subsequent epoch. All queries: CORRECT.

8. PROVERIF FORMAL VERIFICATION RESULTS

Three ProVerif models were constructed, one for each security-critical protocol component. ProVerif is an automatic cryptographic protocol verifier that proves security properties via Horn clause resolution. It reasons about an adversary with the Dolev-Yao capability: the adversary controls the network and can intercept, replay, and modify all messages.

Model	Protocol Component	Property Verified	ProVerif Query	Result
Model 1	MSG-002 ZK Handshake	Zero-knowledge authentication: adversary cannot compute valid proof without ally_channel_fp	query attacker(proof_value).	TRUE — property holds
Model 2	MSG-004 Defensive Leap	Cascade authentication: only nodes with valid MSG-001 chain from origin can verify a Defensive Leap	query attacker(verified_leap).	TRUE — property holds
Model 3	MSG-005 TEE	TEE binding: trust_token	query	All queries

	Management	cannot be computed without TEE access; token is epoch-scoped	attacker(trust_token_epoch_N1).	CORRECT

WHAT PROVERIF PROVES AND DOES NOT PROVE

ProVerif proves security properties of the PROTOCOL MODEL — the abstract description of message exchanges and cryptographic operations. It proves that the protocol design is sound under the Dolev-Yao adversary model. ProVerif does NOT prove the Rust implementation is free of bugs, timing attacks, or side channels. Those properties are addressed by the 683-test suite, Criterion benchmarks, and the red-team evaluation (WP-05). Formal verification and empirical testing are complementary, not substitutes.

9. INTEROPERABILITY

A conforming ASMP/1.0 implementation must:

1. Implement all five message types with the wire formats specified in Sections 3–7
2. Use SHA3-256 (Keccak) for all hash operations and HMAC-SHA3-256 for all message authentication codes
3. Reject any message whose HMAC fails verification
4. Maintain a monotonically increasing cycle_seq per profile
5. Enforce MSG-005 trust_token TTL: reject tokens whose expires_ns is before the current timestamp

A conforming implementation is NOT required to use Rust, to run on x86 hardware, or to use any specific TEE adapter. The HMAC key and TEE quote format are deployment-specific and must be agreed out-of-band between communicating parties.

10. CONCLUSION

ASMP/1.0 provides the network layer that transforms PME into an estate-scale coordinated defence. The five message types cover every inter-node security operation: mutation audit (MSG-001), peer authentication (MSG-002), anomaly propagation (MSG-003), cascade rotation (MSG-004), and management attestation (MSG-005). Three security properties — ZK authentication, cascade authentication, and TEE binding — are formally verified with ProVerif under the Dolev-Yao adversary model.

This specification is intended as an RFC-candidate document. A Go reference implementation and FPGA reference implementation are planned as the next interoperability milestones.