

EchoPulse Multi-Party Key Exchange – Architectural Foundation (v3.0)

1. Overview

This document introduces the architectural foundation for an EchoPulse-based Multi-Party Key Exchange (MPKX) protocol, extending the framework's inherent post-quantum forward secrecy (PQ-FS) and replay resilience to a group communication setting. Traditional multi-party key exchange protocols often face scalability challenges and can struggle with achieving robust forward secrecy in a post-quantum landscape. EchoPulse's MPKX leverages its deterministic, time-bound symbolic graph mutation to enable a scalable, secure, and resource-efficient solution for establishing a shared group key among N participants. The architecture prioritizes decentralization, resilience against single-point compromise, and efficient operation across diverse device classes, from constrained IoT devices to powerful servers.

2. Role Structure and Graph Setup

The EchoPulse MPKX protocol envisions a distributed environment where each participant contributes to the formation of a shared key.

2.1. Participant Roles and Shared Graph Structure

Each participant P_i (where $i \in \{1, \dots, N\}$) holds specific cryptographic elements and contributes to the MPKX process:

- **Public/Private Key Pair (PK_i, SK_i):** Each P_i possesses a long-term EchoPulse public/private key pair. This is used for authentication and signing, but crucially, not for the direct derivation of the session key.
- **Individual Symbolic Path (r_i):** Each P_i generates a fresh, random symbolic path $r_i = (s_{i,1}, s_{i,2}, \dots, s_{i,L})$ of length $L = \text{symbol_path_length}$ for each MPKX session. This path is kept secret by P_i .
- **Local Graph-Time State G_t :** All participants *MUST* deterministically derive the *same* Graph-Time State G_t for a given session. This is achieved using the shared, strictly monotonic forward_entropy_epoch (t), defined as: $G_t = \mu(G_0, t)$ where G_0 is the initial base graph (graph_id), and $\mu(G, t)$ is the deterministic mutation function (mutation_schedule_id). The derivation of t is consistent with File 45 ("EchoPulse PQ Forward Secrecy – TLS Handshake Extension and Enforcement (v3.0)").

2.2. Role-Based Graph Derivation Process

The MPKX process involves participants exchanging partial contributions that enable

all members to derive the final shared key. No single participant holds all the necessary secret material for the final key, decentralizing trust.

MPKX_Init (Initialization Phase):

1. **Epoch Synchronization:** All participants P_i agree on a common $\text{forward_entropy_epoch}(t)$ for the current MPKX session. This can be achieved through:
 - A designated coordinator PC who proposes t (e.g., derived from current time, a session counter, or a random beacon) and signs it, or
 - A distributed consensus mechanism where t is derived from a hash of all participants' initial random nonces, exchanged securely (e.g., via authenticated channels). The latter is preferred for decentralization.
 - Once t is agreed upon and verified, each P_i computes $G_t = \mu(G_0, t)$.
2. **Individual Contribution Generation:** Each participant P_i (for $i=1\dots N$):
 - Generates a fresh, random symbolic path r_i .
 - Selects an initial starting vertex $v_{i,\text{start}} \in V(G_t)$.
 - Traverses G_t using its secret r_i and the Hash Oracle Replacement Model (HORM). The traversal yields a sequence of intermediate HORM outputs $(h_{\text{local},i,j})$ and a final vertex $v_{i,\text{end}}$.
 - For $j=1\dots L$: $(v_{i,j}, h_{\text{local},i,j}) = \text{HORM}(G_t, v_{i,j-1}, s_{i,j}, \text{seed}_H)$.
 - $v_{i,\text{end}} = v_{i,L}$.
 - Computes a *partial key contribution* C_i . This C_i is a cryptographically masked version of $v_{i,\text{end}}$ and/or an aggregated output of $h_{\text{local},i,j}$ values. C_i *MUST NOT* directly reveal r_i or SK_i . It is typically derived via a KDF over the HORM outputs and some public nonce.
 - $C_i = \text{Hash}(\text{v}_{i,\text{end}} \parallel \text{aggregated_HORM_outputs}_i \parallel \text{public_nonce}_i \parallel \text{PK}_i)$
 - Optionally, P_i also includes a "replay token" (derived from t and P_i 's session-specific inputs) to detect message replay at the MPKX layer.
3. **Distribution:** Each P_i broadcasts or securely sends $(C_i, v_{i,\text{start}}, \text{public_nonce}_i, \text{replay_token}_i)$ to all other participants. Each message is signed using P_i 's SK_i to ensure authenticity.

MPKX_Aggregate (Key Derivation Phase):

1. **Collection and Verification:** Each participant P_k collects $(C_j, v_{j,\text{start}}, \text{public_nonce}_j, \text{replay_token}_j)$ from all $j \in \{1, \dots, N\}$.
 - P_k verifies the signature of each C_j using PK_j .

- Pk verifies the replay_token_j against the current session's t and expected values.
- 2. **Deterministic Global Graph Traversal / Aggregation:** All participants *MUST* compute the same final shared secret. This is achieved by defining a deterministic aggregation function or a canonical path traversal order.
 - **Method 1 (Canonical Aggregation):** All participants compute a canonical aggregate of all Cj values. $K_{Group} = KDF-Combine(C1 || C2 || \dots || C_N)$ This method is simple but less reliant on graph traversal for the *final* key.
 - **Method 2 (Sequential Path Traversal):** A deterministic order of initial vertices $v_{j,start}$ and secret paths r_j is defined (e.g., sorted by participant ID). Each participant Pk uses their own secret path r_k to traverse from $v_{k,start}$ to $v_{k,end}$, and then they combine the $v_{j,end}$ values from all other participants in the canonical order. This is complex as it requires secret r_j to be exchanged securely or derived deterministically from some shared secret.
 - **Proposed EchoPulse Method (Chained HORM Traversal):**
 1. Define a canonical order for participants (e.g., by sorted PK_i). Let this be $P(1), P(2), \dots, P(N)$.
 2. The first participant P(1) calculates their full traversal $v_{(1),end}$, $\text{text}\{aggregated_HORM_outputs\}_{(1)}$ using $r(1)$ on Gt.
 3. P(1) sends $v_{(1),end}$ (or a hash thereof) and a *public contribution* to P(2).
 4. P(2) then uses $v_{(1),end}$ as its starting point and traverses Gt using its secret $r(2)$ and HORM. The resulting $v_{(2),end}$ is then passed to P(3), and so on.
 5. This forms a chain: $v_{(1),end} \rightarrow v_{(2),end} \rightarrow \dots \rightarrow v_{(N),end}$.
 6. The final aggregate vertex $v_{final_aggregate}$ is used, along with the combination of all public contributions, to derive the group key.
 7. Each P_i computes their *share* of the final key material KEP_i using their secret r_i and the HORM, similar to how KEP_t is derived in the KEM (File 44).
 8. The final group key is derived by combining all KEP_i using a Group Key Derivation Function (GKDF): $K_{Group} = \text{GKDF}(K_{EP_1} || K_{EP_2} || \dots || K_{EP_N} || \text{MPKX_Context})$ The GKDF must be a robust, order-independent hash or XOR combination (for additively homomorphic properties if possible) to ensure all parties derive the same key. The MPKX_Context includes t, participant IDs, and message hashes.

2.3. Visual Architecture Diagram

Code-Snippet

graph TD

subgraph MPKX_Init

P1[Participant P1] -- Generates r1, derives Gt --> C1(C1, v1_start, PN1)

P2[Participant P2] -- Generates r2, derives Gt --> C2(C2, v2_start, PN2)

PN[Participant PN] -- Generates rN, derives Gt --> CN(CN, vN_start, PNN)

C1 -- Signed_Broadcast --> All_P[All Participants]

C2 -- Signed_Broadcast --> All_P

CN -- Signed_Broadcast --> All_P

subgraph Shared_Context

GO[Initial Graph G0]

mu[Mutation Function μ]

Entropy[forward_entropy_epoch (t)]

G_t[Derived Graph Gt]

G0 -- μ , t --> G_t

Entropy -- binds --> G_t

end

All_P -- Collects & Verifies --> Collected_Contributions[{Ci, vi_start, PN_i}]

end

subgraph MPKX_Aggregate

Collected_Contributions --> GKDF_Input[Input for Group KDF]

GKDF_Input -- Deterministic Ordering/Aggregation --> K_Group[Shared Group

Key K_Group]

end

style C1 fill:#f9f,stroke:#333,stroke-width:2px

style C2 fill:#f9f,stroke:#333,stroke-width:2px

style CN fill:#f9f,stroke:#333,stroke-width:2px

style G_t fill:#ccf,stroke:#333,stroke-width:2px

style K_Group fill:#bfb,stroke:#333,stroke-width:2px

3. Symbolic Key Derivation

The shared key derivation in MPKX is fundamentally rooted in the deterministic, time-bound symbolic graph traversal and the combination of each participant's unique path contribution.

3.1. Key Derivation Logic (KGroup)

Each participant P_k calculates their unique component KEP_k similarly to a 2-party KEM decapsulation. The challenge is to combine these securely.

1. **Derive G_t :** Each P_k independently derives the correct G_t for the session using the agreed-upon `forward_entropy_epoch (t)`.
2. **Verify Peer Contributions:** For each P_j 's received contribution $(C_j, v_{j,start}, \text{public_nonce}_j, \text{replay_token}_j)$, P_k verifies its signature and `replay_token_j`.
3. **Local Key Component Calculation (Iterative HORM Aggregation):**
 - Each P_k uses their secret path r_k to perform a traversal on G_t .
 - The crucial aspect is how P_k integrates the *public* contributions of other parties into their own key component calculation. This can be done by a shared, deterministic HORM aggregation:
$$K_{EP_k} = \text{KDF}_{\text{HORM}}(\text{HORM_Output_Aggregate}_k \parallel \text{Combined_Public_Inputs} \parallel \text{MPKX_Context})$$
 Where `HORM_Output_Aggregate_k` is derived from P_k 's secret traversal. `Combined_Public_Inputs` is a deterministic hash or concatenation of all $C_j, v_{j,start}$, and public_nonce_j values from all participants in a canonical order. This means each participant contributes their "private" path traversal to a common "public" aggregate, which then gets used in a final, deterministic Group KDF.
4. **Group Key Derivation (KGroup):**
 - This step relies on a secure Group Key Derivation Function (GKDF) that aggregates the *individual contributions* from all participants.
 - A common approach is to use a strong hash function over the sorted concatenation of all valid C_j contributions:
$$K_{\text{Group}} = \text{GKDF}(C_{(1)} \parallel C_{(2)} \parallel \dots \parallel C_{(N)} \parallel \text{MPKX_Session_Context})$$
 where $C(j)$ are the canonically ordered (e.g., sorted by participant ID) contributions. `MPKX_Session_Context` includes t , participant IDs, and any other agreed-upon session parameters.
 - This ensures that if all parties correctly contributed and verified, they derive the identical KGroup.

4. Multi-Party Negotiation

The MPKX negotiation extends the principles of TLS 1.3 to a multi-party setting.

4.1. Negotiation Flow

1. Session Setup & Parameter Agreement:

- A designated "session initiator" (or all participants via a pre-agreed mechanism) broadcasts an MPKX session request.
- This request includes proposed `graph_id`, `mutation_schedule_id`, `symbol_path_length`, and other EchoPulse parameters required for Gt derivation.
- Participants respond with their capabilities and agreement.
- The `forward_entropy_epoch` (t) is established. This could be a hash of all participants' random fields from their initial setup messages, combined with a session counter.

2. Contribution Exchange:

- Each P_i generates their r_i and computes C_i (as per `MPKX_Init`).
- P_i sends their signed $(C_i, v_{i,start}, \text{public_nonce}_i, \text{replay_token}_i)$ to all other participants.
- This is typically a broadcast or a multi-cast, requiring secure, authenticated channels.

3. Key Derivation and Verification:

- All participants collect all valid contributions.
- All participants compute `KGroup` using the agreed GKDF.
- An optional "key confirmation" step can be added where participants exchange MACs computed with `KGroup` over a transcript of the MPKX messages.

5. Security Properties

The EchoPulse MPKX is designed to provide robust security properties, building on the strengths of EchoPulse's core KEM.

5.1. Forward Secrecy Across Sessions

- **Epoch-Bound Gt:** Each MPKX session uses a unique Gt, derived from a strictly monotonic `forward_entropy_epoch` (t). The specific Gt used for a past session cannot be re-derived or simulated without the exact session context.
- **Per-Session r_i :** Each P_i generates a fresh, ephemeral r_i for every session.
- **No Centralized Secret:** The group key `KGroup` depends on the combined,

ephemeral contributions (C_i) of all participants. Even if all long-term keys (SK_i) are compromised in the future, the individual contributions (C_i) are transient and do not reveal past r_i values or past KGroup values. This aligns with PQ-FS for 2-party KEMs (File 43).

5.2. Compromise Resilience

- **Individual SKi Leakage:** If one or more individual long-term secret keys (SK_i) are leaked, it compromises the authenticity of future messages signed by that participant but *does not* compromise past KGroup values. The C_i contributions are generated from ephemeral r_i on a specific G_t , which are not recoverable from SK_i .
- **Path r_i Leakage:** If an attacker compromises a single participant P_i and obtains their r_i for a session, they can compute P_i 's *contribution* to KGroup. However, they cannot compute the full KGroup without obtaining the *contributions* of all other participants, which rely on *their* secret r_j and the unique G_t .

5.3. Unlinkability Between Message Flows

- The derivation of t from high-entropy, session-specific random values ensures that each MPKX session is distinct.
- The use of fresh r_i and the dynamic G_t for each session ensures that the generated contributions (C_i) and the resulting KGroup are unlinkable to previous or future sessions. An adversary observing multiple MPKX message flows cannot easily determine if they belong to the same group or individual participants across sessions.

5.4. Deterministic Replay Defense via Epoch Validation

- **Epoch Re-validation:** Each participant, upon receiving MPKX contributions, *MUST* re-validate the `forward_entropy_epoch` (t) and confirm that it matches their independently derived t for the current session.
- **Replay Token:** Each participant P_i includes a `replay_token_i` derived from t and their individual session inputs. Other participants verify this token. If a message is replayed from a previous session, its `replay_token_i` will not match the current session's t , causing rejection. This mechanism is an extension of EchoPulse's core replay resistance.
- **Abort Cases:**
 - If a participant's computed `forward_entropy_epoch` (t) does not match the implicitly agreed-upon t (e.g., detected by a mismatch in a key confirmation

- step or by verification of public hashes), the MPKX session *MUST* abort.
- If any participant's contribution (C_i, \dots) fails signature verification, or if its internal consistency checks (e.g., replay token validation) fail, the MPKX session *MUST* abort.
- If the final KGroup derived by a participant does not match the expected value (e.g., through a key confirmation handshake), the MPKX session *MUST* abort.

6. Distributed Implementation Strategist: Implementation Notes

The EchoPulse MPKX model is designed for flexibility across diverse device classes, leveraging the inherent efficiency of the EchoPulse core.

6.1. Memory and Computational Constraints

- **IoT Devices:** The sub-9KB RAM and sub-15KB ROM footprint of the core EchoPulse KEM makes it feasible for even the most constrained IoT devices to participate in MPKX.
 - Graph G_t is computed locally and deterministically, not stored in its entirety.
 - The r_i path traversal is iterative, requiring minimal memory for intermediate states.
 - The C_i contributions and K_Group are fixed-size cryptographic outputs, not large data structures.
- **Clients/Nodes/Servers:** These devices can leverage hardware acceleration for HORM operations and KDFs, allowing for faster processing of multiple participants' contributions.

6.2. Parallelizable Symbolic Traversal

- The generation of individual r_i and their respective C_i contributions can be performed in parallel by each participant.
- The final GKDF can be implemented in parallel if partial sums or Merkle-tree-like structures are used for aggregation, but the current approach for KGroup (sorted concatenation) is sequential but fixed-time.

6.3. Optional Use of Shared HORM Nodes to Reduce State Size

For large groups or extremely constrained devices, optimization strategies include:

- **Sparse Graph Mutation:** The `mutation_schedule_id` can specify sparse mutations, where only a small fraction of the graph changes at each t , reducing the computational burden of applying $\mu(G, t)$.

- **Pre-computed Graph Snippets:** For highly constrained devices, essential "snippets" or "zones" of GO and common mutation patterns could be pre-computed/cached, rather than the entire graph. The specific graph_id and mutation_schedule_id would dictate which zones are relevant.
- **Shared HORM Instances:** In a network where multiple IoT devices are under a gateway or hub, the gateway could potentially pre-calculate Gt or act as a shared HORM processing node, providing services to the leaves. However, this introduces centralization and impacts compromise resilience.

6.4. Packet Format and Symbol Transport Logic

- **MPKX Contribution Message (MPKX_Contribution):**
 - version: MPKX protocol version.
 - session_id: Unique identifier for the MPKX session (derived from t).
 - participant_id: Identifier of the contributing participant Pi.
 - v_i_start_id: Identifier of Pi's chosen starting vertex.
 - C_i: Pi's cryptographically masked contribution.
 - public_nonce_i: A public nonce from Pi.
 - replay_token_i: A session-specific token for replay detection.
 - signature: Signature over the message using Pi's SKi.
- **Symbol Transport:** The actual symbolic_path_sequence ri is *never* directly transmitted across the network, preserving its secrecy. Only the fixed-size contribution Ci (derived from ri and Gt) is exchanged.

6.5. Role Rotation for Dynamic Joining

- **Adding New Participants:** When a new participant Pnew joins an existing group, a *new* MPKX session **MUST** be initiated.
 - Pnew goes through the MPKX_Init phase, generating their rnew and Cnew.
 - The entire group recalculates a new forward_entropy_epoch (tnew) based on all current members, including Pnew.
 - A new KGroup is derived from the contributions of all N+1 participants. This provides **backward secrecy** (new member cannot decrypt past communications) and **forward secrecy** (compromise of current members does not impact future keys).
- **Participant Departure:** If a participant leaves, a new MPKX session is initiated with the remaining members, recalculating tnew and KGroup based on the reduced set. This provides **forward secrecy** (departed member cannot decrypt future communications).

7. EchoPulse System Integration

The EchoPulse MPKX architecture is deeply integrated with the core EchoPulse framework and its foundational security principles.

- **Full Compatibility with Files 43-45:**
 - **File 43 (PQFS Structure):** The forward_entropy_epoch (t) and the dynamic Graph-Time State ($G_t = \mu(G_0, t)$) are central to MPKX, providing the necessary PQ-FS and replay resistance.
 - **File 44 (PQFS Key Derivation Model):** The fundamental principles of key derivation (path traversal, HORM interaction, KEPT derivation) are extended to the multi-party setting, where individual contributions C_i are essentially "partial" KEPT values or inputs for a final group KDF.
 - **File 45 (TLS Handshake Extension and Enforcement):** The secure and deterministic negotiation of the forward_entropy_epoch (t) for MPKX sessions will leverage similar principles to the 2-party TLS handshake, ensuring all participants compute the identical t.
- **Reuse of Terminology:** Consistent terminology is maintained throughout this document, including $\mu(G, t)$, v_{enc} (conceptually as a part of the C_i derivation), replay token, entropy_epoch, and the KDF_HORM (as the internal hash derivation mechanism within the HORM).
- **Modular Document Structure and Architectural Clarity:** This document adheres to a clear, modular structure to facilitate understanding and integration into the broader EchoPulse dossier.
- **Interoperability:** MPKX contributions are designed to be compact and efficient for network transport, optimizing for various network conditions and device capabilities.

This foundational architecture for EchoPulse MPKX lays the groundwork for secure, scalable, and quantum-resistant group communications, extending the unique benefits of the EchoPulse framework to a broader range of applications.