
EchoPulse Mutation Trace Visualizer: Graphical Debug and Security Analysis Interface

1. Purpose and Scope

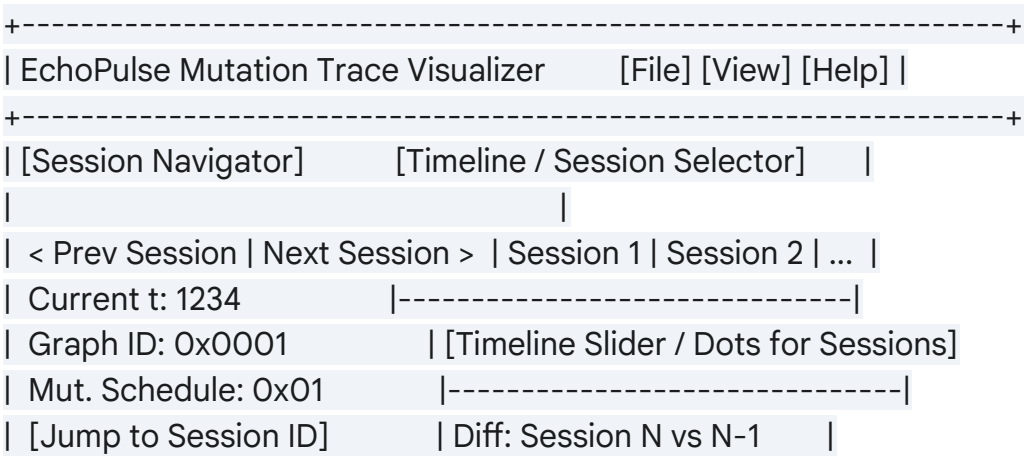
This document specifies the design and functionality of the "EchoPulse Mutation Trace Visualizer," a GUI-based tool intended for developers, researchers, and auditors working with the EchoPulse Key Encapsulation Mechanism (KEM). The primary purpose of this tool is to provide an intuitive graphical interface for understanding, debugging, and analyzing the complex dynamic behavior of the EchoPulse symbolic graph, particularly its deterministic mutation process ($\mu(G,t)$) and symbol-path traversal.

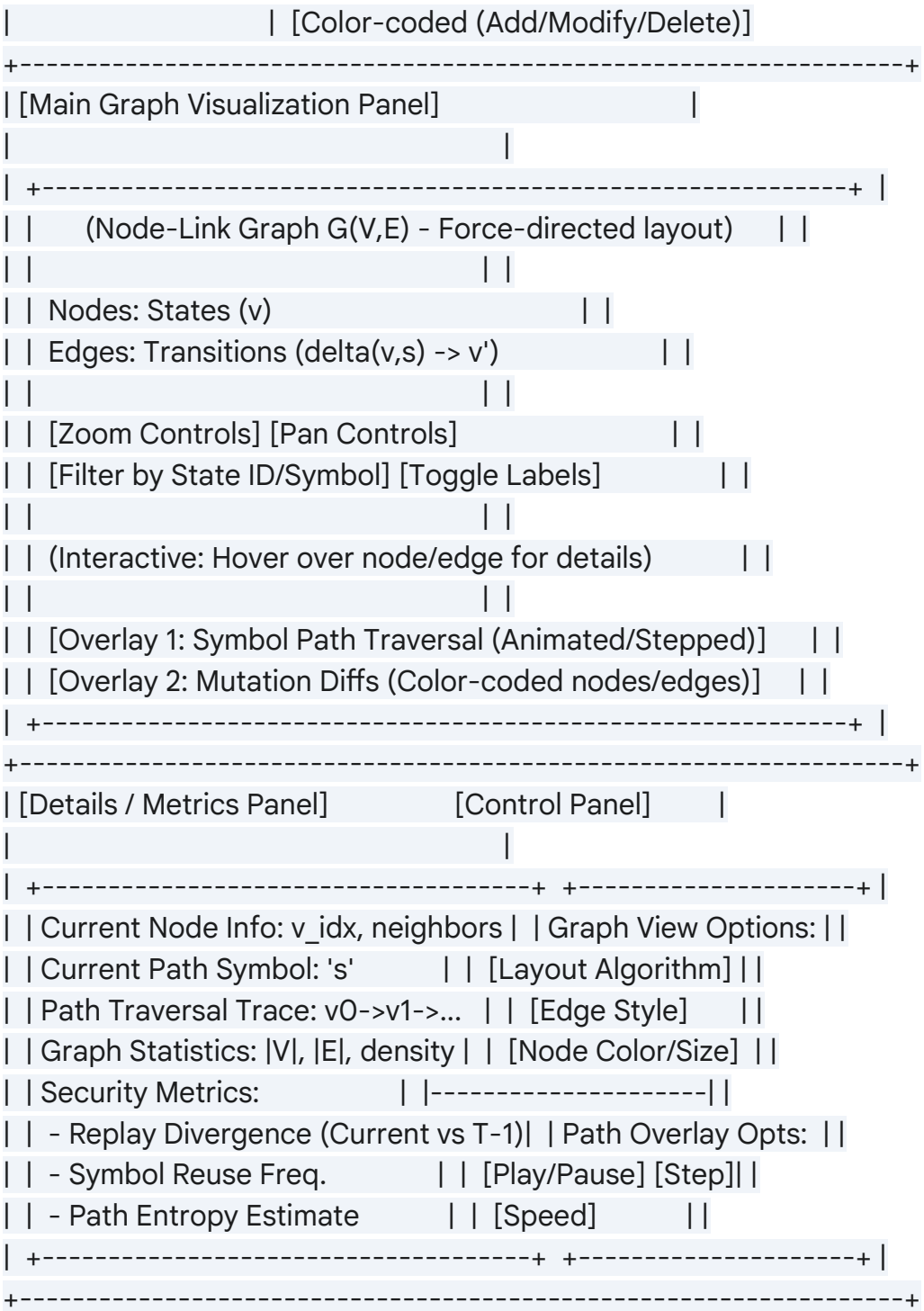
The scope includes detailed UI design, supported input/output formats compatible with EchoPulse internal logging, essential features for debugging, and specialized visualizations for security-relevant metrics.

2. User Interface Layout (GUI Visualization Designer)

The visualizer will adopt a multi-panel layout, providing a holistic view of the EchoPulse session.

2.1. Main Window Layout (Wireframe Concept)





2.2. Key UI Elements and Interactions:

- **Session Navigator:** Allows navigation between different EchoPulse KEM sessions (t values). "Prev/Next Session" buttons, a direct "Jump to Session ID" input, and a display of current session ID (t), graph_id, and mutation_schedule_id.

- **Timeline / Session Selector:** A slider or discrete dots representing each logged session. Selecting a point on the timeline loads the corresponding Gt graph state. This panel also includes a visual "diff" viewer showing changes from Gt−1 to Gt.
- **Main Graph Visualization Panel:**
  - **Graph Display:** Force-directed layout (e.g., Fruchterman-Reingold) of the current Gt. Nodes (vertices) representing states, edges representing transitions.
  - **Zooming/Panning:** Standard controls.
  - **Filtering:** Filter nodes by ID, neighbors, or filter edges by symbol.
  - **Node/Edge Details:** Hovering over a node or edge displays its ID, associated symbol (for edges), and neighboring connections in the Details Panel.
  - **Symbol Path Overlay:** An animated or step-through visualization of the random symbol path r traversing the graph. Nodes visited light up, edges traversed are highlighted. Controls (play/pause, step forward/backward, speed) in the Control Panel.
  - **Mutation Diffs:** When navigating between sessions, the graph visually highlights (color-codes) nodes and edges that have been added, modified, or deleted by the μ(G,t) function between Gt−1 and Gt.
- **Details / Metrics Panel:** Displays contextual information for the currently active graph element (node, edge, path step) and calculates security metrics.
- **Control Panel:** Contains options for graph layout, visualization styles, and symbol path animation controls.

3. Input/Output Formats (Developer Functionality Architect)

The visualizer will consume structured log files and output various visualization and data formats.

3.1. Supported Inputs:

- mutation_trace.json:
  This is the primary input, containing a chronological sequence of EchoPulse KEM sessions. Each entry represents a unique t value and the resulting graph state Gt or the diff necessary to reconstruct Gt from Gt−1.

```JSON
[
 {
  "session_id": 1,
  "t_value": 0,
```

```
  "graph_id": "0x0001",
  "mutation_schedule_id": "0x01",
  "graph_state": {
    "nodes": [{"id": 0}, {"id": 1}, ...],
    "edges": [
      {"from": 0, "symbol": "0x01", "to": 5},
      {"from": 0, "symbol": "0x02", "to": 12},
      ...
    ]
  },
  "initial_pk_ep": "..." // Optional: base64 encoded PK_EchoPulse
 },
 {
  "session_id": 2,
  "t_value": 1,
  "graph_id": "0x0001",
  "mutation_schedule_id": "0x01",
  "mutation_diff": {
    "added_nodes": [],
    "deleted_nodes": [],
    "modified_edges": [
      {"from": 0, "symbol": "0x01", "old_to": 5, "new_to": 7},
      {"from": 10, "symbol": "0x03", "old_to": 20, "new_to": 21},
      ...
    ],
    "added_edges": [],
    "deleted_edges": []
  },
  "ct_ep_received": "...", // Optional: base64 encoded CT_EchoPulse
  "shared_secret_k": "..." // Optional: base64 encoded derived K_EP
 },
 ...
]
```

    - *Note:* The graph_state for t=0 is explicit. For subsequent t values, either the full graph_state or a mutation_diff (preferred for low-footprint logs) is provided. The visualizer will reconstruct Gt internally from Gt−1 and the diff.
- symbolpath.log:

A log file containing the sequence of symbol traversals (r) and the resulting state sequence for each session.

```JSON
[
 {
  "session_id": 1,
  "symbol_path": ["0x01", "0x1A", "0xFF", ...], // Array of symbol values
  "path_trace": [0, 5, 12, ...],              // Array of state IDs visited
  "v_enc": 123                                // Final encapsulated node ID
 },
 {
  "session_id": 2,
  "symbol_path": ["0x0A", "0x0B", ...],
  "path_trace": [0, 2, 8, ...],
  "v_enc": 456
 },
 ...
]
```

3.2. Required Features:

- **Session Explorer (Step-through):** Users can click "Next Session" / "Previous Session" or use the timeline slider to load and visualize Gt for a specific t.
- **Color-Coded Drift Mapping:** Visualize mutation_diff by highlighting affected nodes/edges:
  - Green: Added edges/nodes.
  - Red: Deleted edges/nodes.
  - Blue/Orange: Modified edges (e.g., target node changed for a given state-symbol pair).
- **Graph Interaction:**
  - Node/edge selection to view detailed properties.
  - Toggle display of node IDs, symbol labels on edges.
  - Search for specific state IDs or symbol values.
- **Export:**
  - **PNG/SVG:** Export current graph visualization.
  - **JSON:** Export the loaded graph_state for the current t.
  - **PDF:** Export a report summarizing the session, including graph and metrics.

4. Feature Set Overview

The tool provides distinct capabilities for debugging and security analysis:

- **Debugging Graph Behavior:**
  - Visual inspection of graph structure (Gt).
  - Tracking individual symbol path traversals to verify expected KEM behavior.
  - Pinpointing exact changes caused by μ(G,t) over time.
  - Identifying potential "stuck" states or "dead ends" in the graph evolution.
- **Security Analysis:**
  - **Replay Resistance Visualization:** Show how Gt diverges significantly from Gt−1, validating the mutation's effectiveness in preventing replay attacks by altering the KEM's underlying structure. This can be quantified by comparing the number of changed edges/nodes relative to total graph size.
  - **Symbol Reuse Frequency:** Analyze the distribution of symbols used in the symbol_path (r) over time. This helps ensure high entropy and uniform distribution of r.
  - **Path Entropy per Session:** Estimate the entropy of the derived symbol path r for each session, providing a quantitative measure of its unpredictability. This can be visualized as a metric over the timeline.
  - **State Traversal Distribution:** Visualize how frequently different states are visited across multiple symbol_path traversals, helping identify potential biases or hotspots.
- **Optional: Overlay Cryptographic Context:** For each session, display the PKEP, CTEP, and derived KEP as text (Base64/Hex), allowing cross-referencing with other test vectors or KEM implementations.

5. Security Metric Mapping (Security Metrics Visualizer)

The visualizer will provide quantitative and qualitative insights into EchoPulse's security properties.

- **Replay Resistance (Divergence After μ):**
  - **Visualization:** The "Diff" view in the timeline panel visually highlights changes. The Details / Metrics Panel quantifies this.
  - **Metric:** Percentage of edges changed between Gt−1 and Gt. $\text{Divergence} = \frac{|\Delta E| + |\Delta V|}{|E_{t-1}| + |V_{t-1}|} \times 100\%$ (where $|\Delta E|$ is number of changed edges, $|\Delta V|$ is number of changed nodes, and $|E_{t-1}|$ and $|V_{t-1}|$ are edges/nodes in previous graph).
  - **Interpretation:** Higher divergence confirms effective mutation, reducing the

likelihood of successfully replaying a CTEP from a past session.

- **Symbol Reuse Frequency:**
  - **Visualization:** A histogram or bar chart in the Details / Metrics Panel showing counts of each symbol value (0-255) appearing in the symbol_path for the current session or aggregated across selected sessions.
  - **Metric:** Frequency counts for each symbol.
  - **Interpretation:** A flat distribution (uniform usage) indicates good randomness, while spikes suggest potential bias or predictability in the symbol generation process for r.
- **Path Entropy per Session:**
  - **Visualization:** A numerical value and potentially a bar chart in the Details / Metrics Panel. Over time, a line graph of entropy values could be displayed.
  - **Metric:** Shannon entropy of the symbol_path r for the current session, $H(r) = -\Sigma_{s \in \Sigma} P(s) \log_2 P(s)$, where P(s) is the observed probability of symbol s. Normalized entropy (relative to max possible entropy for path length) can also be shown.
  - **Interpretation:** High entropy confirms that the random path r is indeed unpredictable and provides sufficient cryptographic strength.
- Optional: Cryptographic Context Overlay:
  Display the PK_EP, CT_EP, and K_EP values in a dedicated text area. This helps connect the visual graph behavior directly to the cryptographic artifacts of the KEM.

6. Extension Modules (Specification Integrator)

The visualizer could be extended with advanced functionality:

- **Mutation Simulator:** A module that allows users to manually define graph mutation rules and see their effect on a base graph, providing a sandbox for designing and testing new μ functions.
- **Adversary Simulation:** A module that simulates an adversary trying to guess r or venc by analyzing visible graph patterns and historical mutation data. This could demonstrate the effectiveness of the mutation process.
- **Performance Profiler Integration:** Overlay performance metrics (e.g., mutation computation time, path traversal time) onto the timeline view, correlating performance with graph complexity.

7. Final Format Considerations

This specification targets an open-source implementation.

- **Backend:** Python is suitable, leveraging libraries like NetworkX for graph structures, matplotlib or plotly for plotting, and PyQt or Tkinter for GUI.
- **Frontend (Web-based):** For a web-based tool, D3.js or React with a graph visualization library (e.g., react-force-graph) would be excellent choices, with a Python/Node.js backend for data processing.
- **Compatibility:** The tool relies heavily on the mutation_trace.json and symbolpath.log formats. Strict adherence to these JSON schemas is critical for interoperability with EchoPulse internal logging.

This detailed specification provides a roadmap for developing a powerful debugging and analysis tool for EchoPulse, essential for its secure and efficient deployment.