Symbol Entropy Drift Plot Generator for EchoPulse: Visualizing Symbol Reuse and Path Deviation Over Time

1. Purpose

This document specifies a Python-based data analysis and visualization tool designed to assess the quality and security of the symbolic paths generated within the EchoPulse Key Encapsulation Mechanism (KEM). The tool focuses on visualizing how the distribution and usage of symbols (from the EchoPulse alphabet Σ) evolve across multiple KEM sessions, helping to identify potential entropy degradation, symbol reuse, or deterministic path deviation. Such patterns could indicate vulnerabilities to replay attacks or side-channel leakage, directly impacting the security posture of EchoPulse in post-quantum deployments, especially with SGPU (Symbolic Graph Processing Unit) requirements.

2. Symbol Entropy Analysis (Symbol Entropy Analyst)

The core of the tool involves computing several per-session statistical metrics for the symbols in the encapsulated path r.

2.1. Per-Session Statistics:

- **Symbol Frequency Distributions:** For each session, compute the absolute and relative frequencies of each symbol (0-255) used in its symbol_path. This forms the basis for entropy calculation and drift detection.
- **Shannon Entropy per Session (**$H(r)$**):** For each session's symbol_path $r=(s_1,s_2,\ldots,s_L)$, where L is the symbol_path_length: $H(r)=-\sum_{s\in\Sigma}P(s)\log_2 P(s)$ where $P(s)$ is the observed probability of symbol s in the current session's path. The maximum possible entropy for a path of length L over an alphabet of size $|\Sigma|=256$ is $L\times\log_2 256=L\times 8$ bits.
- **Normalized Entropy:** $H_{norm}(r)=H(r)/(L\times 8)$. This provides a percentage of ideal entropy.
- **Drift Metric: Symbol Reuse vs. Session 0 Baseline:** For each symbol s, calculate a "reuse score" relative to the first session (session_id = 0 or t=0). This can be the Kullback-Leibler (KL) divergence or Jensen-Shannon divergence between the current session's symbol frequency distribution and the baseline distribution. A simpler approach is to track how many top-N most frequent

symbols from session 0 are *also* top-N most frequent in subsequent sessions, or simply the sum of absolute differences in probabilities. $Drift_s = |P_t(s) - P_0(s)|$ (Absolute difference for symbol s at time t vs. time 0) $TotalDrift = \sum_{s \in \Sigma} |P_t(s) - P_0(s)|$

- **Symbol Overlap (Binary Presence):** For each session, create a binary vector indicating the presence (1) or absence (0) of each symbol (0-255) in its symbol_path. This can be used for set-based comparisons.

2.2. Degradation Detection:

The tool will detect and log potential issues:

- **Entropy Degradation:** A significant drop in $H(r)$ or $H_{norm}(r)$ over time (e.g., dropping below a configurable threshold, or a sustained negative trend).
- **Overuse of Subsets:** If a small subset of symbols consistently appears with high frequency across many sessions, indicating non-uniform randomness. This would be visually apparent as bright horizontal lines on the heatmap.

3. Python Plot Generation (Python Plot Generator)

The visualization will primarily consist of a heatmap, optionally augmented with a line plot for entropy.

3.1. Libraries:

- pandas: For data loading and manipulation.
- numpy: For numerical computations, especially for entropy.
- matplotlib.pyplot: For core plotting functionalities.
- seaborn: For high-quality heatmaps.

3.2. Drift Plot Design:

- **Type:** Heatmap (e.g., seaborn.heatmap).
- **X-axis:** session_id (representing discrete sessions).
- **Y-axis:** Symbol Index (0 to 255). This provides a granular view of each symbol's behavior.
- **Color Scale:**
  - **Option 1 (Frequency):** The intensity of the color for each (session_id, symbol_index) cell represents the frequency (or probability) of that symbol in that specific session's path. A perceptually uniform colormap (e.g., viridis, magma) is recommended.
  - **Option 2 (Presence):** Binary colors (e.g., black/white) to indicate if a symbol

was present or absent in the path for that session.

- **Optional: Entropy Overlay:** A secondary line plot (e.g., matplotlib.axes.Axes.twinx) on the same figure, showing the Normalized Entropy for each session. This allows correlation between overall entropy and specific symbol usage patterns.

3.3. Rendering Logic (Pseudocode/Function Signatures):

Python

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import argparse
import json
import os

def load_symbol_data(input_path: str) -> pd.DataFrame:
    """Loads symbol path data from CSV or JSON."""
    if input_path.endswith('.csv'):
        # Assuming CSV has 'session_id' and 'symbol_path' as string like "0x01,0x0A,0x1F"
        df = pd.read_csv(input_path)
        df['symbol_path'] = df['symbol_path'].apply(lambda x: [int(s, 0) for s in x.split(',')])
    elif input_path.endswith('.json'):
        # Assuming JSON format: [{"session_id": 1, "symbol_path": [1, 10, 31, ...]}, ...]
        df = pd.read_json(input_path)
    else:
        raise ValueError("Unsupported input file format. Use .csv or .json")
    return df

def calculate_symbol_stats(df: pd.DataFrame) -> pd.DataFrame:
    """
    Calculates per-session symbol frequency, entropy, and baseline drift.
    Returns a DataFrame suitable for plotting.
    """
    all_sessions_data = []
    symbol_alphabet_size = 256 # Assuming 8-bit symbols
```

```python
    # Calculate baseline (Session 0) frequency distribution
    baseline_path = df[df['session_id'] == 0]['symbol_path'].iloc[0] if 0 in df['session_id'].values else None
    baseline_counts = np.zeros(symbol_alphabet_size)
    if baseline_path:
        for symbol in baseline_path:
            baseline_counts[symbol] += 1
    baseline_prob = baseline_counts / len(baseline_path) if baseline_path else np.ones(symbol_alphabet_size) / symbol_alphabet_size # Handle empty baseline gracefully

    for index, row in df.iterrows():
        session_id = row['session_id']
        symbol_path = row['symbol_path']
        path_length = len(symbol_path)

        counts = np.zeros(symbol_alphabet_size)
        if path_length > 0:
            for symbol in symbol_path:
                if 0 <= symbol < symbol_alphabet_size: # Ensure symbol is within bounds
                    counts[symbol] += 1

        probabilities = counts / path_length if path_length > 0 else np.zeros(symbol_alphabet_size)

        # Calculate Shannon Entropy
        # Avoid log(0) for symbols not present
        non_zero_probs = probabilities[probabilities > 0]
        shannon_entropy = -np.sum(non_zero_probs * np.log2(non_zero_probs)) if len(non_zero_probs) > 0 else 0.0
        max_entropy = np.log2(symbol_alphabet_size) * path_length if path_length > 0 else 0.0 # For the path length
        normalized_entropy = shannon_entropy / max_entropy if max_entropy > 0 else 0.0

        # Calculate Total Drift from baseline
        total_drift = np.sum(np.abs(probabilities - baseline_prob))
```

```python
        # Prepare data for heatmap: one row per session, 256 columns for symbols
        session_data = {'session_id': session_id,
                        'shannon_entropy': shannon_entropy,
                        'normalized_entropy': normalized_entropy,
                        'total_drift': total_drift}
        for i in range(symbol_alphabet_size):
            session_data[f'symbol_{i}'] = probabilities[i] # Store probability for heatmap

        all_sessions_data.append(session_data)

    return pd.DataFrame(all_sessions_data)

def plot_drift_heatmap(df_stats: pd.DataFrame, output_path: str, title: str,
                       annotate_anomalies: bool = False, replay_risks: list = None):
    """
    Generates the symbol entropy drift heatmap with optional entropy line plot.
    """
    fig, ax1 = plt.subplots(figsize=(15, 8))

    # Heatmap for symbol frequencies
    heatmap_data = df_stats.set_index('session_id').filter(regex='^symbol_').T
    sns.heatmap(heatmap_data, cmap="viridis", annot=False, fmt=".3f", linewidths=.01,
ax=ax1,
                cbar_kws={'label': 'Symbol Frequency (Probability)'})

    ax1.set_title(title)
    ax1.set_xlabel("Session ID")
    ax1.set_ylabel("Symbol Index (0-255)")
    ax1.set_yticks(np.arange(0, 256, 16)) # Set y-ticks for better readability
    ax1.set_yticklabels(np.arange(0, 256, 16))

    # Optional: Overlay Entropy Line Plot
    ax2 = ax1.twinx()
    ax2.plot(df_stats['session_id'], df_stats['normalized_entropy'], color='red', linestyle='--',
marker='o', label='Normalized Entropy')
    ax2.set_ylabel("Normalized Entropy", color='red')
    ax2.tick_params(axis='y', labelcolor='red')
    ax2.legend(loc='upper left')
```

```
    # Highlight replay risks
    if replay_risks:
        for session_id in replay_risks:
            ax1.axvline(x=session_id - df_stats['session_id'].min(), color='red', linestyle=':',
linewidth=2,
                        label=f'Replay Risk (Session {session_id})' if session_id == replay_risks[0] else
"")
        # Add a single legend entry for replay risks
        handles, labels = ax1.get_legend_handles_labels()
        handles_ax2, labels_ax2 = ax2.get_legend_handles_labels()
        ax2.legend(handles + handles_ax2, labels + labels_ax2, loc='upper left')


    plt.tight_layout()
    plt.savefig(output_path)
    plt.close()

# Main script execution
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description="EchoPulse Symbol Entropy Drift Plot
Generator.")
    parser.add_argument("--input", type=str, required=True,
                        help="Path to the input CSV or JSON symbol trace file.")
    parser.add_argument("--output", type=str, default="drift_plot.png",
                        help="Path to save the output drift plot (e.g., drift_plot.png).")
    parser.add_argument("--entropy-stats", type=str,
                        help="Optional: Path to save per-session entropy statistics as JSON.")
    parser.add_argument("--highlight-top-n", type=int, default=5,
                        help="Highlight top N most frequent symbols (not yet implemented in plot).")
    parser.add_argument("--threshold-overlap", type=float, default=0.90,
                        help="Threshold for session overlap to flag as replay risk.")

    args = parser.parse_args()

    # Load and process data
    try:
        df_symbols = load_symbol_data(args.input)
```

```python
    df_stats = calculate_symbol_stats(df_symbols)

    # Replay Risk Detection (simple example: high overlap with previous session)
    replay_risk_sessions = []
    if len(df_stats) > 1:
        for i in range(1, len(df_stats)):
            current_session_probs = df_stats.iloc[i].filter(regex='^symbol_').values
            prev_session_probs = df_stats.iloc[i-1].filter(regex='^symbol_').values

            # Using cosine similarity as an overlap metric for simplicity
            # This is a basic example; more robust replay detection would use
            # specific hashes of the path and check against a database.
            dot_product = np.dot(current_session_probs, prev_session_probs)
            norm_current = np.linalg.norm(current_session_probs)
            norm_prev = np.linalg.norm(prev_session_probs)
            if norm_current > 0 and norm_prev > 0:
                overlap = dot_product / (norm_current * norm_prev)
                if overlap > args.threshold_overlap:
                    replay_risk_sessions.append(df_stats.iloc[i]['session_id'])


    # Generate plot
    plot_drift_heatmap(df_stats, args.output,
                       "EchoPulse Symbol Frequency Drift and Normalized Entropy",
                       replay_risks=replay_risk_sessions)
    print(f"Generated drift plot: {args.output}")

    # Save entropy statistics
    if args.entropy_stats:
        stats_to_save = df_stats[['session_id', 'shannon_entropy', 'normalized_entropy',
'total_drift']].to_dict(orient='records')
        with open(args.entropy_stats, 'w') as f:
            json.dump(stats_to_save, f, indent=2)
        print(f"Generated entropy statistics: {args.entropy_stats}")

  except Exception as e:
    print(f"Error: {e}")
```

5. Replay Risk Visual Marker (Replay Risk Visual Marker)

The plots will visually flag potential replay risks based on entropy degradation and symbol reuse patterns.

- **High-Frequency Symbols:** Visually, these will appear as consistently bright (hot) horizontal lines across many sessions in the heatmap. This highlights symbols that are used disproportionately often.
- **Deterministic Repeats:** While direct detection of deterministic repeats in paths requires comparison of full paths, the drift plot can reveal the *patterns* that make such repeats possible. If the distribution of symbols becomes highly skewed or collapses to a very small subset, the likelihood of deterministic repetition increases. Visually, this means many symbols remain "cold" (unused) while a few become consistently "hot."
- **Session Pairs with >90% Overlap (Configurable Threshold):** The script will (as a basic example in pseudocode) identify sessions where the symbol frequency distribution is extremely similar to the previous session. These sessions will be marked on the plot (e.g., with a vertical red line or shaded background). This indicates a lack of sufficient mutation/randomness between successive sessions, potentially making a ciphertext from one session replayable in another.
- **"Red Flags" on Zones of Potential Replay Risk:**
  - **Low Entropy Regions:** Areas on the plot where the overlaid entropy line drops significantly will be visually identifiable.
  - **High Overlap Regions:** Sessions flagged by the overlap metric will have distinct visual indicators (e.g., a dashed red vertical line on the session ID).
  - **Consistent Hot Spots:** Symbols that remain highly frequent across many sessions despite expected mutation and randomness.

6. Output and Integration Design (Output and Integration Designer)

The tool is designed for ease of use in a command-line environment and for integration into automated analysis pipelines.

6.1. Input:

- **Format:** CSV or JSON trace logs.
- **File Naming:** Expected to be provided via --input argument, e.g., symbol_trace.csv or symbol_trace.json.
- **Contents:** Each entry must contain session_id and symbol_path (as an array of

integers in JSON or a comma-separated string of hex/decimal values in CSV).

6.2. Output:

- **drift_plot.png:** The primary visual output, saved as a PNG image by default. Configurable via --output.
- **Optional entropy_stats.json:** A JSON file containing the calculated session_id, shannon_entropy, normalized_entropy, and total_drift for each session. This allows programmatic access to the calculated metrics. Configurable via --entropy-stats.

6.3. CLI Usage:

The script is invoked directly from the command line:

Bash

```bash
# Basic usage: Generate drift plot from CSV
python drift_plot.py --input my_ep_trace.csv --output my_drift_plot.png

# Generate plot and save entropy statistics from JSON
python drift_plot.py --input my_ep_trace.json \
            --output plots/ep_drift.png \
            --entropy-stats plots/ep_entropy_stats.json

# Adjust overlap threshold for replay risk flagging
python drift_plot.py --input my_ep_trace.csv \
            --output my_drift_plot_strict.png \
            --threshold-overlap 0.95
```

6.4. Visual Clarity:

- **Tooltips/Legend:** The heatmap will have a clear color bar indicating symbol frequency. The overlaid entropy plot will have its own legend. Axis labels will be clear.
- **Session ID Mapping:** If session_ids in the data are not sequential, the X-axis will accurately represent them.
- **Readability:** Font sizes, line thicknesses, and color choices will be optimized for

readability.

This comprehensive specification ensures the "Symbol Entropy Drift Plot Generator" is a robust, reproducible, and valuable tool for analyzing the critical randomness and evolutionary properties of EchoPulse's symbolic KEM paths.

Quellen

1. https://github.com/Ioana-P/NER_classifier_in_pytorch