APPENDIX A
ARTIFACT APPENDIX

A. Description & Requirements

This artifact appendix provides details on how to reproduce the results presented in OptiMix. OptiMix leverages two types of environments to evaluate its methodologies: (1) an analytical setting and (2) a simulation setting. In the analytical scenario, each technique is evaluated using datasets from Nym or RIPE, assessing the effectiveness of OptiMix methods in reducing latency while quantifying the associated anonymity loss. The methods evaluated include LONA (node adjustments), multiple routing strategies (LAR, GWR, GPR, and SSR), LBA (load balancing), and CRG (cover traffic generation). In contrast, the simulation setting—implemented in *Python* using the *SimPy* library—models the full end-to-end flow of communication in mixnets. Within the simulation, clients generate messages that are forwarded through mixnodes according to OptiMix routing strategies. Each mixnode applies mixing processes to anonymize and forward packets, with delays modeled using empirical measurements from Nym or RIPE to capture realistic inter-node transmission latency.

The artifact comprises approximately 20K lines of Python code and reproduces all results presented in the main body of the paper.[1]

1) How to access: The artifact repository is available via a persistent DOI: https://doi.org/10.5281/zenodo.15827489, and also on GitHub: https://github.com/OptiMixnet/OptiMix.

2) Hardware dependencies: Note that the artifact includes precisely the same configurations and settings presented in OptiMix. The only difference is that the number of iterations has been scaled down so that the artifact can run on standard systems with 16 GB RAM and 50 GB of disk space.

The estimated runtimes are based on executing the artifact on a system with the following specifications:

- 64-bit CPU: Intel® Core™ i7-10850H @ 2.70 GHz (1 core used)
- Physical Memory (RAM): 16 GB

3) Software dependencies: Additionally, the server must have Python 3.8–3.10 installed (not Python ≥ 3.11). We emphasize the importance of adhering to the specified hardware and software dependencies to ensure that the experiments complete within the expected time limits.

4) Benchmarks: OptiMix relies on latency and geographical data from two datasets: (i) **Nym**, which captures latency between nodes in the deployed Nym mixnet, and (ii) **RIPE**, which provides latency measurements among globally distributed nodes for broader network topologies. Both datasets are included in the repository.

B. Artifact Installation & Configuration

After setting up a system with Python version 3.8 or higher, one can execute the artifact by first installing the required dependencies using the script specified in `dependencies.txt`, which is provided in the repository bundled with the code. Upon installing the dependencies, running `Main.py` with the arguments explained below will generate the results, which will be saved in the `Figures` or `Tables` directories.

C. Major Claims

The major claims of OptiMix are summarized as follows:

- (C1): LATENCY PATTERN
  Fig. 2 illustrates the latency incurred in mixnets when applying different routing strategies. As the value of τ increases, latency also increases across all strategies. Overall, LAR, GWR, and GPR exhibit comparable latency, while SSR consistently incurs the highest latency.
- (C2): ENTROPY PATTERN
  Fig. 3 shows that increasing τ results in higher entropy across all routing strategies. In particular, LAR exhibits very low entropy for $\tau \leq 0.6$, while SSR consistently achieves the highest entropy overall.
- (C3): IMPACT OF LOAD BALANCING
  Fig. 4 evaluates the effect of applying the load balancing algorithm (LBA) to the routing strategies. It demonstrates that both latency and entropy increase across all values of τ for cascade and stratified topologies—compared to the results in Figs. 2 and 3.
- (C4): OVERALL PERFORMANCE
  Fig. 5 highlights that when OptiMix is applied to cascade and stratified topologies, with or without LBA, the overall performance—measured as the ratio `Entropy/Latency`—is maximized when using either GPR or GWR strategies across all values of τ.
- (C5): MIXNODE ADVERSARY
  Fig. 8 presents the fraction of fully compromised paths (FCP) under an adversary controlling a subset of mixnodes. The FCP is shown as a function of τ and the adversarial budget $\frac{f}{N}$, where f denotes the number of compromised nodes. The results indicate that increasing τ or decreasing $\frac{f}{N}$ reduces the FCP.
- (C6): COVER ROUTING
  Fig. 9 demonstrates that applying the cover routing generation (CRG) method—especially with a slight increase in the cost parameter θ—results in higher entropy and lower FCP across both stratified and cascade topologies.

D. Evaluation

This section details the set of experiments conducted to support the main claims presented in the artifact appendix. Executing these experiments produces results stored in the `Results`, `Figures`, or `Tables` directories. Additionally, we explain how to run scripts to regenerate specific results corresponding to figures or tables presented in the main body of the paper.

1) Experiment (E_0) [Setup] (≤ 5 min): This experiment prepares all required datasets and auxiliary files. It should be run once before executing any other experiment.

[1]As the paper was accepted at the time of artifact submission, the AEC reviewed and validated all results reported in the main body of the camera-ready version.

[Preparation and Execution] Run the following command with the `Input` argument set to `0`:

```
python3 Main.py
```

[Results] The required files will be generated and stored in the `Results` folder or the root directory of the artifact.

2) Experiment (E_1) [Figures 2–5] (≤ 5 min): This experiment supports claims **C1–C4** by generating baseline latency and entropy results.

[Preparation and Execution] Run the following command with the `Input` argument set to `1`:

```
python3 Main.py
```

[Results] The results will be saved in the `Figures` folder as: Fig. 2a–2d, Fig. 3a–3d, Fig. 4a–4d, and Fig. 5a–5d.

[Note] You may encounter the following warnings during execution, which are safe to ignore:

```
RuntimeWarning: Mean of empty slice.
  out=out, **kwargs

invalid value encountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
```

3) Experiment (E_2) [Figure 8] (≤ 5 min): This experiment evaluates the impact of mixnode adversaries, supporting claim **C5**.

[Preparation and Execution] Run the following command with the `Input` argument set to `2`:

```
python3 Main.py
```

[Results] The output will include Fig. 8a–8d, saved in the `Figures` folder.

4) Experiment (E_3) [Figure 9] (≤ 5 min): This experiment evaluates the cover routing strategy and supports claim **C6**.

[Preparation and Execution] Run the following command with the `Input` argument set to `3`:

```
python3 Main.py
```

[Results] The results will be saved in the `Figures` folder as Fig. 9a–9d.

5) Experiment (E^*) [All Others] (≤ 40 min): This experiment allows the user to generate any figure or table shown in the main body of the paper, regardless of whether it is linked to a claim.

[Preparation and Execution] Refer to Table I below for the corresponding `Input` argument value and run:

```
python3 Main.py
```

[Results] Figures will be saved in the `Figures` folder. Tables will be saved in the `Tables` folder as PDF files. However, in cases where a LaTeX installation is not available, table data may be printed directly in the terminal.

TABLE I
MAPPING OF INPUT ARGUMENTS TO SPECIFIC FIGURES AND TABLES.

Experiment	Input	Experiment	Input
Fig. 2	22	Fig. 3	33
Fig. 4	44	Fig. 5	55
Fig. 6	66	Fig. 7	77
Fig. 8	88	Fig. 9	99
Tab. 1	100	Tab. 2	200
Tab. 3	300	—	—

[Note] You may encounter the following warnings while running experiments related to Fig. 2–4 or Tab. 3. These warnings do not affect the correctness of the results and can be safely ignored:

```
RuntimeWarning: Mean of empty slice.
  out=out, **kwargs

invalid value encountered in scalar divide
  ret = ret.dtype.type(ret / rcount)
```

E. Execution Time

The execution time of our experiments—whether supporting core claims or generating specific figures or tables—should not exceed 40 minutes under the specified hardware and software configurations, including the default settings in `config.py`. The total runtime for all experiments sequentially should remain under half a day. Further, note that parallel execution across multiple cores can significantly reduce the overall runtime, potentially allowing all experiments to complete in under an hour.

F. `config.py` Parameters

The `config.py` file contains default initialization parameters, including the number of iterations, which are chosen to balance accuracy and runtime. By default, the number of iterations is set to 5, which we found sufficient to capture trends aligned with the results reported in the paper. For experiments used in the paper, a higher iteration count (e.g., 400+) was applied to ensure robustness. Please note that we do not guarantee execution times for configurations that differ from those specified in the artifact.