ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

A Artifact Appendix

A.1 Abstract

This artifact is separated into two parts: **simulations** (Sections 6 and 7 on the paper) and **evaluations** (Sections 8 and 9).

The simulations part assumes a machine with Python3.8 and the libraries *scipy*, *numpy*, *lightning-utils*, *networkx*, *matplotlib*, and *seaborn*. This part runs standalone simulations and was used to generate Figures 4, 5, 6, 7, and 10 on the paper. In practice, to reduce running time, we ran most of the simulations using Slurm on an internal cluster.

On the part of the evaluation, we implemented the Twilight system in 4 components: smart contract (Solidity), enclave (C++, SGX), relay (Python), and the evaluations manager (Python). The enclave and relay parts were run on an Azure VM. The tests were written using *pytest*, used Ganache as a local blockchain, and the compiler *solcjs* to compile the contract. In order to create fully reproduceable results, this evaluation part can be executed only in Azure cloud environment. The manager creates the relevant resources in the cloud and executes the system according to the evaluation experiments. To run this part, we assume a machine with Python3.8, the libraries *paramiko* (to establish SSH connection to the machines) and *matplotlib*, *seaborn* to generate the plots, the Pythonic Azure SDK, credentials with admin permissions (to create resources such as VMs, NICs, IPs, etc.), and enough quota in Azure to create these machines. To create Figures 8 and 9 in the paper, we used 6 VMs with the type *Standard_DC1s_v2* (3 in each region: *eastus* and *northeurope*).

A.2 Artifact check-list (meta-information)

- **Algorithm:** We implemented Algorithm 1 from the paper (Appendix, Section A) inside the SGX (file *Enclave/tree.cpp*), and Algorithm 2 (Appendix, Section B) inside the smart contract (file *smart_contract/channel.sol*).
- **Compilation:** Most of our code is written in Python. The part of the enclave is written in C++17 and could be compiled by running *make SGX_MODE=HW SGX_PRERELEASE=1* in the root directory. The smart contract can be compiled using any solidity compiler, we used *solcjs* without any flags.
- **Hardware:** The evaluation uses an Azure VM of type *Standard_DC1s_v2*, with SGX-1. The local code runs in Python and has no hardware requirements.
- **Experiments:** In the evaluation part, we created a line topology of 6 machines: Alice, 4 relays, and Bob. For every throughput value that has been tested, we executed a command on Bob's machine to initiate the given amount of payments every second. Then, after 10 seconds, we queried Alice's machine for the number of finished payments (to get the throughput) and for the duration of the payment (to get the latency).
- **How much disk space required (approximately)?:** Negligible. Less than 1GB should be sufficient.
- **How much time is needed to prepare workflow (approximately)?:** Creating and preparing all the cloud resources is being executed once, and take less than an hour overall.
- **How much time is needed to complete experiments (approximately)?:** Each data point in Figures 9 and 10 should be executed separately and takes around 20 minutes (can be controlled by lowering the number of repetitions. We used a default of 20 repetitions).
- **Publicly available (explicitly provide evolving version reference)?:** The Github repository is: `https://github.com/saart/Twilight`.
- **Code licenses (if publicly available)?:** None.
- **Data licenses (if publicly available)?:** We used the Lightning Network topology which was queried (using a standard CLI command `https://github.com/lightningnetwork/lnd/blob/593962b788753768661582d11221f32ebf7dbe67/cmd/lncli/commands.go#L1515`) from a Lightning node. This is publicly available.
- **Workflow frameworks used?:** We used Python and Azure's SDK to manage the experiments (initiate and teardown machines), FastAPI (`https://fastapi.tiangolo.com/`) as the communication framework between the relays, and Pistache (`https://github.com/pistacheio/pistache`) as the communication framework between the relay and the enclave.
- **Archived (explicitly provide DOI or stable reference)?:** On the paper we used tag: `https://github.com/saart/Twilight/tree/USENIX-Security-22`.

A.3 Description

A.3.1 How to access

Publicly available at: `https://github.com/saart/Twilight`

A.3.2 Hardware dependencies

None (run on machines with specific requirements on the cloud).

A.3.3 Software dependencies

Python3.8 with the libraries *scipy*, *numpy*, *networkx*, *matplotlib*, and *seaborn*. For the evaluation part, Azure SDK is also required. Moreover, we assume network connectivity, and in particular the ability to run Azure CLI command and establish SSH sessions to Azure's VMs.

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

To install the Azure VM follow the description in: `https://github.com/saart/Twilight#how-to-install-on-a-new-azure-confidetial-computing-machine`. To install the local machine: Install Python3.8 with `https://www.python.org/downloads/`, install setuptools using *pip install setuptools==58.0.0*, install azure cli using *pip install azure-cli==2.19.0* and install the rest of requirements using *pip install -r simulations/requirements.txt*. Then, create permissions from your Azure profile using `https://docs.microsoft.com/en-us/azure/developer/python/sdk/authentication-overview` (make sure that you can authenticate using python, e.g. by running the Pythonic command *get_client_from_cli_profile(ComputeManagementClient)*).

A.5 Experiment workflow

The simulation workflow is standalone per Figure:

- Figure 4 can be reproduced using *simulations/distinct_routes.py*
- Figures 5 and 6 can be reproduced using *simulations/visualizations/efficiency_privacy_tradeoff.py*
- Figure 7 can be reproduced using *simulations/noise_simulations/liquidity_distribution.py* and then *simulations/noise_simulations/success_rate.py*
- Figure 10 can be reproduced using *simulations/visualization/adoption.py*

The evaluation workflow is managed using the script *simulations/manage_tests.py*.The general flow is:

1. Create the machines in the relevant regions (*eastus* and *northeurope*).
2. Start/Restart the machines, which starts two processes: the relay and the enclave as system services. This step also refreshes the existing machines between different experiments.
3. Build the P2P and channels topology. I.e. query for the names of the relays, and execute *register* to create the edges.
4. Execute a command on Bob's machine, that initiates a repetitive thread that starts payments according to the given route and the given desired throughput.
5. Query Alice's machine on the payments that have been finished in the last few seconds, and store the throughput (number of finished payments) and the latency (duration of each payment).

Steps 2-5 are re-executed repetitively: both to evaluate the same experiment again, and to evaluate using different parameters (number of issued payments, route length).

In order to plot the results and reproduce Figures 8 and 9, use the file *simulations/draw_evaluation_figures.py*.

A.6 Evaluation and expected results

Our main claim in the paper is that Twilight is a valid solution to the probing attack of off-chain networks.

This claim is backed by this artifact that presents both simulations and evaluations of the system and its properties. Every one of the Figures 3-10 from the paper is backed with the code that is presented in this artifact.

To reproduce the results of the simulations part (Figures 3-7 and 10), follow the description in Section A.5, and run each file to generate the corresponding figure.

To reproduce the results of the evaluation part (Figures 8 and 9), first connect to your azure environment using the bash command *az login*, authenticate in the opened browser, and run the evaluation using the script *simulations/manage_tests.py* from the directory *simulations/*. Then, draw the plots using *draw_evaluation_figures.py*.

The results from both parts should plot the same graphs that we presented on the paper. This is possible that the results will vary, therefore for each plot on the paper we included error bars that should present the range of the variation.

A.7 Experiment customization

The topology of the evaluation is flexible. Although we presented on the paper only a line-topology, we included in the file *simulations/manage_tests.py* more possible topologies that evaluate different use-cases. The different use cases that we also examined are: changes in the throughput over time, building a topology based on the topology of the Lightning network, and two routes that intersect in the middle (X topology).

A.8 Notes

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.