USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

# Encrypted Access Logging for Online Accounts: Device Attributions without Device Tracking

Carolina Ortega Pérez and Alaa Daffalla, *Cornell University;*
Thomas Ristenpart, *Cornell Tech*

https://www.usenix.org/conference/usenixsecurity25/presentation/ortega-perez

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.


# USENIX Security '25 Artifact Appendix: Encrypted Access Logging for Online Accounts: Device Attributions without Device Tracking

Carolina Ortega Pérez* Cornell University
Alaa Daffalla* Cornell University
Thomas Ristenpart Cornell Tech

## A Artifact Appendix

### A.1 Abstract

Our artifact implements the protocol described in Section 4 of our paper. The artifact is composed of a `simulator.py` file, that can be used to run the end-to-end login and log retrieval algorithms from our protocol and a README explaining how to do so. Additionally, we have a `server2.py`, a `client2.py` and an `encryptor2.py` files, which represent the server, client, and encryptor components, respectively, of our protocol. At a high level, the server sends data related to the encrypted log of actions to the client, and the client sends requests to the encryptor to perform cryptographic operations. As mentioned throughout the paper, the protocol is agnostic to the authentication mechanism.

Moreover, our prototype includes files `encryptor_entry_size.py` and `algorithms_payload_size.py` that provide the size measurements (in bytes) for individual components, enough to analyze the payload of the remaining functions. We report the results from the prototype in Section 7 of our paper.

### A.2 Description & Requirements

#### A.2.1 Security, privacy, and ethical concerns

Our prototype fetches the serial number of the device on which it is running. However, the program runs locally and does not send this information anywhere or stores it persistently. Besides this, our prototype does not involve any human subjects or personal data. All account and authentication data is synthetic, for example we use synthetic randomly generated cookies, session identifiers, and user and relying party information.

#### A.2.2 How to access

Our artifact is publicly available in GitHub at https://github.com/alaadaff/csal_code and Zenodo at https://zenodo.org/records/14737179. .

*Authors contributed equally to this work.

#### A.2.3 Hardware dependencies

Our code is developed for and compiled on a MacOS device. We performed our experiments on a MacBook Pro with an M1 CPU and 8 GB of RAM running Sonoma v.14.6. We have not tested it on other operating systems or devices.

The only OS-dependent code is the function that retrieves the device's serial number, which utilizes macOS's `system_profiler`. This command does not require root privileges. The function is implemented in the `def getSerial()` function within the `encryptor2.py` file.

To adapt this code for other operating systems, replacing the function's implementation with an equivalent command should be sufficient. For example:

- On Windows, the serial number can be fetched using the `wmic bios get serialnumber` command.
- On Linux, the serial number can be obtained using the `dmidecode -s system-serial-number` command.

However, we note that both the Windows and Linux commands may require administrator or root privileges. We have not tested the code on these platforms.

#### A.2.4 Software dependencies

Our artifact is written in Python and requires installing the cryptography library version 42.0.5 and the pyhpke library version 0.5.3.

#### A.2.5 Benchmarks

None.

### A.3 Set-up

To set up our prototype, all that is needed is downloading the code from Zenodo.

### A.3.1 Installation

Our artifact can be downloaded from https://zenodo.org/records/14737179. It is implemented in Python and requires the cryptography and pyhpke libraries. These can be installed by running `pip install -r requirements.txt`.

### A.3.2 Basic Test

Running `python3 simulator.py -e login-no-smuggle -i 1` should open two new terminals simulating a server and client respectively. It should run without any error, and print some performance values.

## A.4 Evaluation workflow

### A.4.1 Major Claims

**(C1):** The payload for a login without key smuggling between the server and the client is 1,687 B. The response back to the client takes 3,224 B. The round-trip for one login takes 7ms (CPU time for both server and client to complete operations). This is reported in Section 7, in the login evaluation.

**(C2):** A log can be retrieved from the server and decrypted locally to display the log entries. The decrypted log contains the serial number of the device on which it is running. This is reported in Section 7, in the log retrieval evaluation.

**(C3):** A log retrieval payload from the server takes 2,482 B (no smuggling) and 2,631 B with smuggling. This is reported in Section 7, in the log retrieval evaluation.

**(C4):** A re-encryption request for a single session to be re-encrypted takes 2,653 B. This is reported in Section 7, in the re-encryption evaluation.

**(C5):** An entry in the encryptor database takes 2,263 B. This is reported in Section 7, in the login evaluation.

### A.4.2 Experiments

**(E1):** *[login (no smuggle)] [approx. 1 human-minutes + 1 compute-minute]: Runs a login flow, sends data from the server to the client to the encryptor and back. This measures the login without key smuggling.*
**How to:** Run the command `python3 simulator.py -e login-no-smuggle -i 1`.
**Results:** The experiment opens two terminals, which print the CPU time and the payload size.

**(E2):** *[log retrieval] [approx. 1 human-minutes + 1 compute-minute]: Runs a log retrieval flow, sends data from the server to the client to the encryptor, where the log is decrypted and printed.*
**How to:** Run the command `python3 simulator.py -e history -i 2`.
**Results:** The experiment opens two terminals, which retrieves the log showing decrypted log entries.

**(E3):** *[log retrieval] [approx. 1 human-minutes + 1 compute-minute]: Computes the payload size for a single log retrieved from server.*
**How to:** Run the command `python3 log_retrieval2.py`.
**Results:** The experiment prints the size of a single log returned from the server during log retrieval.

**(E4):** *[other algorithm payloads] [approx. 1 human-minutes + 1 compute-minute]: Computes the payload size for re-encryption and action.*
**How to:** Run the command `python3 algorithms_payload_size.py`.
**Results:** The experiment prints the size of the payload at the different stages of the re-encryption and action.

**(E5):** *[encryptor storage] [approx. 1 human-minutes + 1 compute-minute]: Computes the size of the different components in an encryptor entry.*
**How to:** Run the command `python3 encryptor_entry_size.py`.
**Results:** The experiment prints the size of each element on each row of an encryptor entry.

## A.5 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.