USENIX Security '26 Artifact Appendix: LPG: Raise Your Location Privacy Game in Direct-to-Cell LEO Satellite Networks

Quan Shi[1], Liying Wang[2]

A Artifact Appendix

A.1 Abstract

Multi-tenant direct-to-cell (D2C) Low Earth Orbit (LEO) satellite networks pose significant risks to users' location privacy by linking Mobile Network Operator (MNO)- managed identities with Satellite Network Operator (SNO)- visible locations. We present LPG, the first protocol-layer framework that decouples identity from location while maintaining essential functions like billing and policy in resource-constrained orbital environments.

This artifact provides the implementation of the LPG protocol, including the source code for the User Equipment (UE), Satellite Network Operator (SNO), and Mobile Network Operator (MNO) components. The paper's evaluation involves a real in-orbit LEO satellite. Since this hardware is physically inaccessible for public or reviewers, this artifact focuses on the functional correctness of the protocol logic. We also provide raw telemetry logs extracted from the satellite flight to substantiate the paper's specific hardware claims.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

We make our research artifacts accessible in `https://zenodo.org/records/17906146` (Public Version) and `https://zenodo.org/records/18428501` (Restricted Version).

A.2.3 Hardware dependencies

The LPG protocol can be evaluated on standard commodity hardware.

- Processor: The paper's evaluation results (latency, power, temperature) were collected on the BUPT-3 satellite payload (8-core ARM processor, 16 GB RAM). An ARM64 Linux environment (e.g., Raspberry Pi 4/5 or an ARM-based VM) is recommended.

A.2.4 Software dependencies

The artifact is tested on Ubuntu 20.04 LTS and Ubuntu 22.04 LTS. It requires a hybrid build environment for C++, Rust, and Go.

- OS: Linux (Ubuntu 20.04/22.04).
- Compilers & Build Tools:
  - **C++** Compiler supporting C++17 (e.g., `g++` or `clang++`).
  - **Rust** and `cargo` (required for the SCAC module integration).
  - **Go** (version 1.18+, required for ZKLP bridge compilation).
  - `CMake` ($\geq$ 3.10) and `Make`.
  - `git`.
- System Libraries:
  - OpenSSL development libraries (`libssl-dev`).
  - GMP development libraries (`libgmp-dev`).
  - `pkg-config`.
- Python: Python 3 and Jupyter Notebook are required to regenerate the plots using `scripts/plot.ipynb`.

A.2.5 Benchmarks

The artifact includes both benchmarks for protocol verification and real-world telemetry datasets collected from the BUPT-3 LEO satellite in-orbit deployment.

- Protocol Benchmarks: The `lpg_main` (from `LPG-code`) generates synthetic user identities and tokens to execute the full LPG protocol lifecycle, including UE and SNO unlinkable token generation (DualToken), Offline Authentication and Key Agreement (OAKA), Zero-Knowledge Location Proof (ZKLP) generation that enables session management, and Scalable Collaborative Aggregate Claiming (SCAC).

- Satellite Telemetry Datasets: The Results directory contains raw execution logs and hardware telemetry (power consumption, temperature) collected from the BUPT-3 LEO satellite (COSPAR ID 2025-103F). These datasets correspond to the evaluation results reported in Section 5 of the paper.

A.3 Set-up

A.3.1 Installation

The artifact is designed for a standard Linux environment (Ubuntu 20.04+ recommended). The build process requires a hybrid environment for C++, Rust, and Go.

1. Install System Dependencies

```
sudo apt update
sudo apt install -y libssl-dev pkg-config \
    libgmp-dev
```

2. Update and Integrate Submodules

```
git submodule update --init --recursive
sh LPG-code/appy-integration.sh
```

3. Build and Run

```
cd LPG-code
mkdir build && cd build
cmake ..
make
./lpg_main
```

Upon success, the `lpg_main` binary is generated in the `build` directory.

A.3.2 Basic Test (Liying)

`lpg_main` integrates simple tests for basic functionalities. The expected successful output is shown in Figure 1.

A.4 Evaluation workflow

A.4.1 Major Claims

The major claims of our paper are that LPG provides a functional, privacy-preserving protocol framework for Direct-to-cell D2C LEO networks, operating within the constraints of satellite hardware. Since the computational tasks running on SNO are conducted in a real space environment, the results you obtain when running the code on the server may differ from those presented in the paper.

(C1): Efficient Token Registration. LPG's DualToken mechanism achieves practical generation times ($\sim$ 33ms for $\mathsf{Tok}_{\mathsf{UE}}$) and storage overhead comparable to commercial SIM cards. This is validated by experiment (E1) and discussed in §5.3.1 (Figures 12 (b)(c)).

```
(base) ➜  LPG-code ./build/lpg_main
===== LPG System Integration Test =====

------ Testing Dual Token Mechanism ------
Dual Token Mechanism Test: PASSED

------ Testing OAKA Protocol ------
OAKA Protocol Test: PASSED

------ Testing Session Management Table (SMT) ------
SMT Test: Session Creation PASSED
SMT Test: Session Update PASSED
SMT Test: Session Termination PASSED

------ Testing ZKLOC ------
[Go] Compiling Circuit (this happens only once)...
17:01:47 INF compiling circuit
17:01:47 INF parsed circuit inputs nbPublic=4 nbSecret=2
17:01:47 INF building constraint builder nbConstraints=25486
[Go] Running Setup (generating keys)...
ZKLOC Test: Initialization PASSED
Input Lat: 0.691891, Lng: 0.709619, Resolution: 0, I: 1, J: 0, K: 1
17:01:50 DBG constraint system solver done nbConstraints=25486 took=17.903625
17:01:50 DBG prover done acceleration=none backend=groth16 curve=bn254 nbConstraints=25486 took=102.4445
ZKLOC Test: Proof Generation PASSED
17:01:50 DBG verifier done backend=groth16 curve=bn254 took=1.987291
ZKLOC Test: Proof Verification PASSED

------ Testing SCAC (Groth16 ZK-SNARK) ------
Start:   Groth16::Generator
··Start:   Constraint synthesis
··End:     Constraint synthesis .....................................................885.834µs
··Start:   Inlining LCs
··End:     Inlining LCs .............................................................2.815ms
··Start:   Constructing evaluation domain
··End:     Constructing evaluation domain ..........................................23.459µs
··Start:   R1CS to QAP Instance Map with Evaluation
····Start:   Evaluate Lagrange coefficients
····End:     Evaluate Lagrange coefficients .......................................605.791µs
··End:     R1CS to QAP Instance Map with Evaluation .................................1.468ms
··Start:   Compute G2 table
··Start:   Compute G2 table
··End:     Compute G2 table ........................................................1.592ms
··Start:   Calculate B G2
··End:     Calculate B G2 ..........................................................2.220ms
··Start:   Compute G1 window table
··End:     Compute G1 window table .................................................1.280ms
··Start:   Generate the R1CS proving key
····Start:   Calculate A
····End:     Calculate A ...........................................................903.791µs
····Start:   Calculate B G1
····End:     Calculate B G1 ........................................................617.375µs
····Start:   Calculate H
····End:     Calculate H ...........................................................1.662ms
····Start:   Calculate L
····End:     Calculate L ...........................................................1.018ms
··End:     Generate the R1CS proving key ...........................................5.118ms
··Start:   Generate the R1CS verification key
··End:     Generate the R1CS verification key ......................................362.750µs
··Start:   Convert proving key elements to affine
··End:     Convert proving key elements to affine ..................................780.125µs
End:     Groth16::Generator .....................................................17.725ms
Start:   Groth16::Prover
··Start:   Constraint synthesis
··End:     Constraint synthesis .....................................................704.625µs
··Start:   Inlining LCs
··End:     Inlining LCs .............................................................2.547ms
··Start:   R1CS to QAP witness map
··End:     R1CS to QAP witness map .................................................4.290ms
··Start:   Compute C
··End:     Compute C ...............................................................5.026ms
··Start:   Compute A
··End:     Compute A ...............................................................1.782ms
··Start:   Compute B in G1
··End:     Compute B in G1 .........................................................1.350ms
··Start:   Compute B in G2
··End:     Compute B in G2 .........................................................3.330ms
··Start:   Finish C
··End:     Finish C ................................................................1.500µs
End:     Groth16::Prover .....................................................19.333ms
SCAC Test: ZK-SNARK Proof Verification PASSED
===== LPG System Integration Test Finished ====
```

Figure 1: Expected bash output when running `./lpg_main`

(C2): Lightweight Attachment (OAKA). The OAKA protocol successfully performs anonymous authentication with acceptable latency ($\sim$ 140ms extra latency) and low communication overhead. This is validated by experiment (E1) and discussed in §5.3.2 (Figures 14 (a)).

(C3): User-Configurable Location Disclosure (ZKLP). The ZKLP mechanism supports variable location resolutions ($\mathsf{res} \in \{0..15\}$) with consistent proof sizes ($\sim$ 430 bytes) and verification times. This is validated by experiment (E1) and discussed in §5.3.3 (Figure 13).

(C4): Scalable Settlement (SCAC). The SCAC protocol efficiently aggregates claims, with server overhead and constraint counts scaling linearly with the number of sessions. This is validated by experiment (E2) and discussed in §5.3.4 (Figures 15 (a)(b)).

(C5): Satellite Viability. The on-board components of LPG operate within the strict power (< 40W) and temperature ($< 62^{\circ}$C) budgets of LEO satellites. This is supported by

```cpp
// In LPG-code/main.cpp
// N=8
const size_t num_sessions = 8;
SCAC_SNO sno(num_sessions);
SCAC_MNO mno(sno.GetZKKeys(), sno.GetPoseidonParams());

// length of the vectors: K=4
std::vector<uint64_t> individual_data_usages = {100, 150, 50, 200};
std::vector<uint64_t> individual_session_ids = {1001, 1002, 1003, 1004};
```

Figure 2: Adjusting parameters $\mathcal{N}$ and $\mathcal{K}$

the in-orbit telemetry logs analyzed in experiment (E3) and discussed in §5.2 and §5.3.4 (Figures 15 (d)(e)(f)). This involves measurements of the satellite hardware and is not included in the logs output by our `lpg_main` code.

A.4.2 Experiments

We describe the experiments to validate the functional correctness of the LPG protocols (E1, E2) and to audit the real-world satellite execution traces (E3).

(E1): Protocol Functional Verification: Validates the cryptographic correctness and logic flow of Registration, OAKA, and ZKLP (Claims C1, C2, C3).
Preparation: Ensure all dependencies (C++, Rust, Go) are installed and the project is built successfully using `cmake` as described in the Installation section.
Execution: Run the main test.
Results: The program will sequentially execute Token Generation, Offline Authentication and Key Agreement (OAKA), and ZKLP Proof Generation.

- **Expected Outcome:** The console output should display "PASSED" for each module.
- **Note:** Execution times reported on the evaluator's local machine will differ from the ARM-based satellite COTS device results reported in the paper, but the logical correctness and computational complexity of ZKLP will be maintained.

(E2): Settlement Scalability Test: Validates the scalability of the SCAC aggregation protocol (Claim C4).
Preparation: Requires the same build environment as E1.
Execution: Run the main test.
Results: The script simulates the aggregation of multiple sessions.

- **Expected Outcome:** The program completes aggregation without error. Evaluators should observe that the processing time and constraint generation follow the linear trend described in §5.3.4, validating the scalability claim.

(E3): Satellite Telemetry Audit: Validates the real-world deployment and environmental impact claims (Claim C5) by inspecting artifacts generated in orbit.
Preparation: Locate the `Results/Satellite/` directory in the artifact package.
Execution: This is a data inspection task. Reviewers should examine the provided log files:

- Execution Logs: e.g., `at/arm-at-*.log` (OAKA execution).
- Telemetry Logs: e.g., `...temp-power.log` (Hardware sensors).

Results: Compare the timestamps in the execution logs with the telemetry logs.

- **Expected Outcome:** The logs confirm that the experiments were conducted on the BUPT-3 satellite platform. The power and temperature readings in the logs should match the ranges reported in Table 2 and Figure 14 (b) and Figure 15 (d)(e)(f) of the paper (e.g., Power $<$ 40W), substantiating the system's viability.

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/usenixsec2026/.