Project 2 - Design & Details

Project Title: SplitSmart
Project Type: Project 2.7 - Designing an end-to-end cryptography solution to protect your data application from attacks
Team: Gagan Yalamuri and Yathish Naraganahalli Veerabhadraiah

Application context:

The proposed system is a small, networked expense-splitting service called SplitSmart, intended for a fixed group of users such as roommates or friends. Each user records shared expenses by specifying who paid, how much was paid and a short description. The backend maintains an append-only ledger of these expense entries and derives running balances from this ledger so that participants can see roughly who owes whom.

Threat model and assumptions:

Attacker capabilities:

- Full control over the network path between clients and the server.
- Ability to capture and replay old messages.
- Read/write access to the backend storage.

Assumptions:

- Client devices and long-term secret keys are not compromised.
- Users are already authenticated to their client (login/UI authentication is out of scope).
- The number of users is small and fixed (e.g., a known group of participants).

Security Goals:

- Confidentiality - Expense details (payer, amount, description) must be hidden from eavesdroppers.
- Integrity - Any modification of messages in transit or entries in storage must be detectable.
- Origin authentication - The system must verify which user created each ledger entry and detect impersonation.
- Replay protection - Replayed old messages, even if valid once, must be detected and rejected.
- Tamper-evident history - Deleting, inserting, or modifying stored entries must leave visible evidence.

Client Component:

- Minimal interface (CLI or simple web UI) to:
    - Start a secure session with the server.
    - Create and submit expenses.

    - Request the ledger and balances.
- Handles:
    - Running the authenticated key exchange at session start.
    - Constructing and signing expense records.
    - Encrypting and authenticating messages before sending.

Server Component:

- Maintains:
    - Registered users and their public keys.
    - Long-term server keys.
    - Per-user counters for replay protection.
    - The ledger and corresponding hash chain.
- On each incoming message:
    - Verifies the channel MAC.
    - Decrypts the payload.
    - Verifies the user's signature.
    - Check the counter for replay.
    - Verifies and updates the hash chain before committing the entry.

Storage:

- Uses a simple file or lightweight database to store:
    - Ledger entries and their hashes.
    - User metadata and counters.
- On startup, recomputes and checks the hash chain to detect offline tampering.

Design choices:

- We intentionally use a three-layer authentication structure (signed handshake for the channel, per-entry user signatures for actions, and MACs for transport) so that different attack surfaces are covered by different keys and algorithms.
- We design the ledger as a hash-chained, receipt-backed log, giving users cryptographic evidence of their recorded actions and making any server-side tampering with history detectably inconsistent.
- We also apply simple application-level key evolution, periodically deriving fresh session keys from existing ones, to limit the impact of any long-lived key compromise.