Overview

This project implements a custom TOTP-based multi-factor authentication system in Python. The system consists of a TOTP library (following RFC 6238) and a Flask server to simulate login workflows. It enforces user-bound OTPs and login rate limit.

Methodology

The TOTP class generates OTPs using:
– A shared secret unique to each user.
– A 30-second time step (parameterized).
– HMAC-SHA1 (parameterized for other hash algorithms, too) for generating OTPs based on the secret and the counter (derived by the current time and the timestep).
– Dynamic truncation to extract a code from the hash.

For the verification there is a parameter of how many adjacent time windows to check for the case of time desynchronization.

The shared secret should have the same byte size as the output of the hash function, as mentioned in RFC 2104, section 3.

The algorithm is explicitly described in RFC 6238 section 4, RFC 4226 section 5.
Key method: `_generate_otp(counter)`

Flask Server

– `/register`: Registers users, generates and shares a secret (in base32).
– `/login`: Authenticates the user using username and OTP.

Security measures

– The shared secret is transmitted over HTTP, which is not secure.
– OTPs are stored per user.
– Rate-limiting: 1 login attempt per 5 seconds (prevents brute-force attacks).

Client Application

A lightweight client communicates with the server to:
– Register new users and get their secrets.
– Attempt login using OTPs that it can generate.

Moreover, a client external device simulator has been implemented, which, given the secret, continuously shows the current OTP.

Results

– OTPs are bound to individual users via per-user secret keys.
– OTPs are valid only for a limited time: 30 seconds (at most) by default.
– A rate limit on login attempts ensures resistance to brute-force attacks.

Limitations

– This project is a simple TOTP demo, which is why it intentionally skipped the first authentication step via the traditional static password, and focused on the second step of the OTP authentication, assuming that the first step has already been completed successfully.
– All user and OTP data is stored in-memory; persistence is lost on restart.
– There's no mobile authenticator or QR integration; OTPs are managed manually.
– The project runs over HTTP by default, which would need to be replaced with HTTPS for production use.