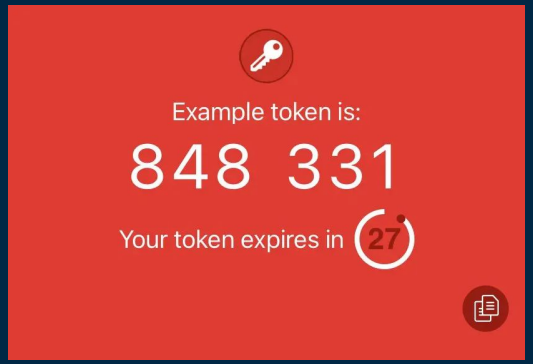
TOTP Manager

Chenxi Liu (21096794d)
Haolin Zhang (21107569d)
Tianji Huang (21099573d)
Xu Le (21096101d)

CONTENTS

Introduction to TOTP



What is TOTP?

- A dynamic, time-sensitive one-time password generation algorithm.
- Based on a shared secret key and the current time.
- Widely used for two-factor authentication (2FA).

How Does It Work?

- Generates a unique, short-lived code (usually 30 seconds).
- Relies on synchronized clocks between client and server。

Key Uses of TOTP:

- Enhances security for online accounts and applications.
- Commonly used in banking, enterprise systems, and personal account protection.
- Compatible with authenticator apps like Google Authenticator and Microsoft Authenticator.

Advantages:

- Easy to implement and use.
- Reduces risks of unauthorized access.

Challenges in current TOTP products and our solutions

Challenges

Lack of Multi-Device Synchronization

- Difficult to seamlessly sync TOTP across multiple devices.
- Causes inconvenience for users managing multiple platforms.

Closed-Source Nature

- Limited transparency in security protocols.
- Users cannot verify the integrity of the underlying code.

Lack of Transparency

- Users have little insight into how their data is handled.
- Raises concerns about trust and security.

Our solution

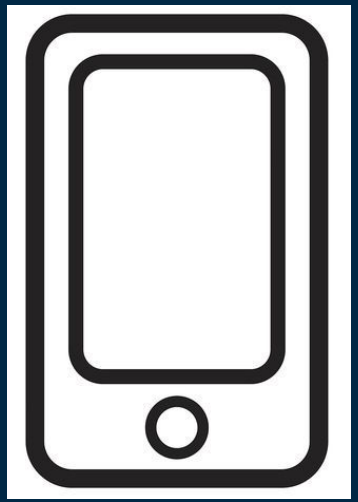
Addressing the Gaps:

- **Multi-Device Synchronization**: Effortless syncing across all devices.
- **Open-Source Platform**: Full transparency and community-driven improvements.
- **Enhanced Transparency**: Clear processes and secure data handling practices.

Objective:

- To provide a **user-friendly, secure, and transparent** TOTP solution.
- Empower users with **control, trust, and convenience** in managing their authentication needs.

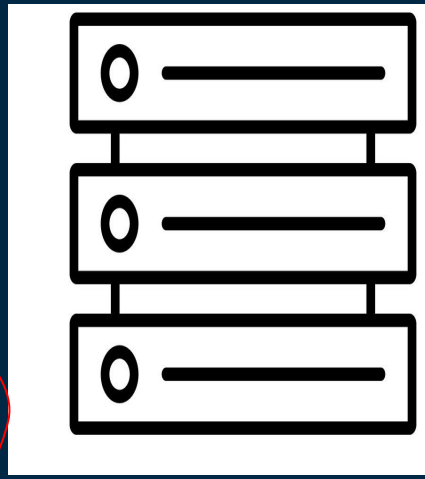
Protocol design-upload



Mobile 1

1. Define the user secret k
2. Request upload by sending request to /upload interface
3. Generate 1024-bit uid and 128-bit skk and store uid in redis
4. Send back server-defined uid and skk
5. Store skk and uid
6. Send encrypted message and uid
7. Store the received message in redis with refer to uid

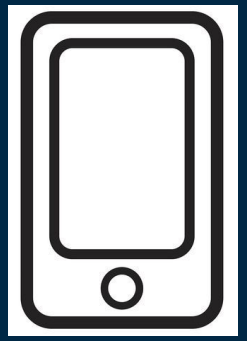
Push



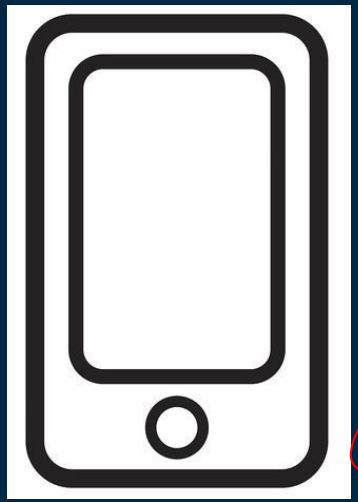
Server

Protocol design-Sync

1. Show Sync QR code



Mobile 1



Mobile 2

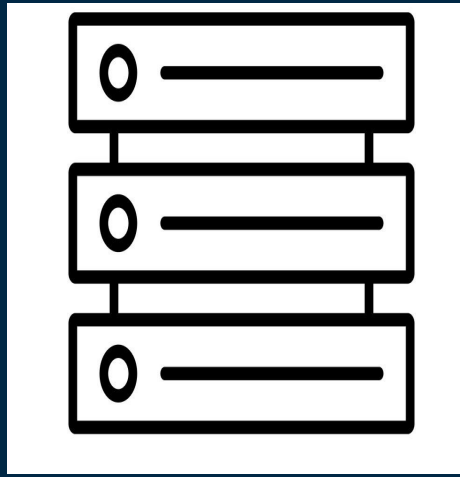
2. Scan QR Sync QR code and get Uid

3. Send Sync request by /Sync interface, must contain uid

4. Send back corresponding encrypted data

5. Parse and show TOTP information

Fetch



Server

Protocol design-Consistency ensure

In the worst-case scenario, multiple devices may upload data simultaneously, leading to potential data overwrites between devices.

To address this issue, we have defined an additional mechanism. Since the server processes incoming requests sequentially, the uploaded data is handled in a specific order. When all devices complete their push and perform a fetch, they will always see the last processed data.

However, instead of immediately overwriting the device's data with the fetched data, we perform a merge operation. **This merge combines the fetched data from the server with the local data, ensuring consistency across multiple devices.**

Technical details

Biometrics

- Utilizes Android Biometric API for user authentication.
- Supports fingerprint or device credentials for secure access.
- Ensures unauthorized access to the application and sensitive data is prevented.

HTTPS

- Ensures secure communication between client and server using SSL/TLS.
- Provides end-to-end encryption for data in transit.
- Employs custom SSL configurations for flexibility during development.

Key Storage

- Leverages Android KeyStore for secure and hardware-backed encryption key storage.
- Protects sensitive credentials and keys from unauthorized access.
- Prevents keys from being exported, ensuring secure isolation.

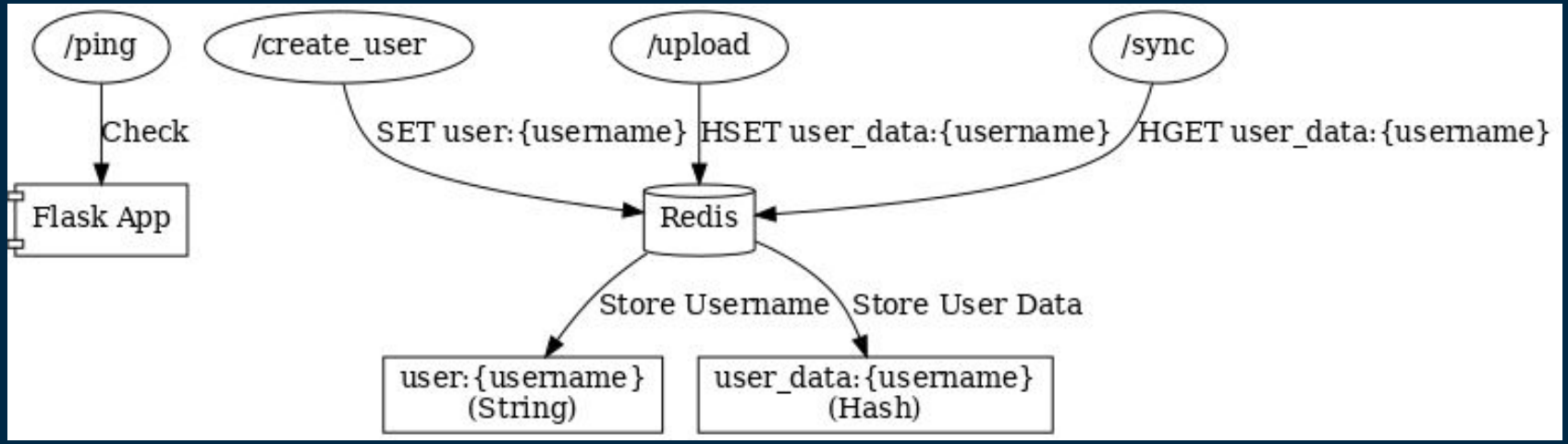
Android Cipher

- Implements AES encryption in GCM and CBC modes for robust data security.
- Maintains confidentiality and integrity of stored and transmitted data.
- Encryption keys securely derived through SHA-256 hashing mechanisms.

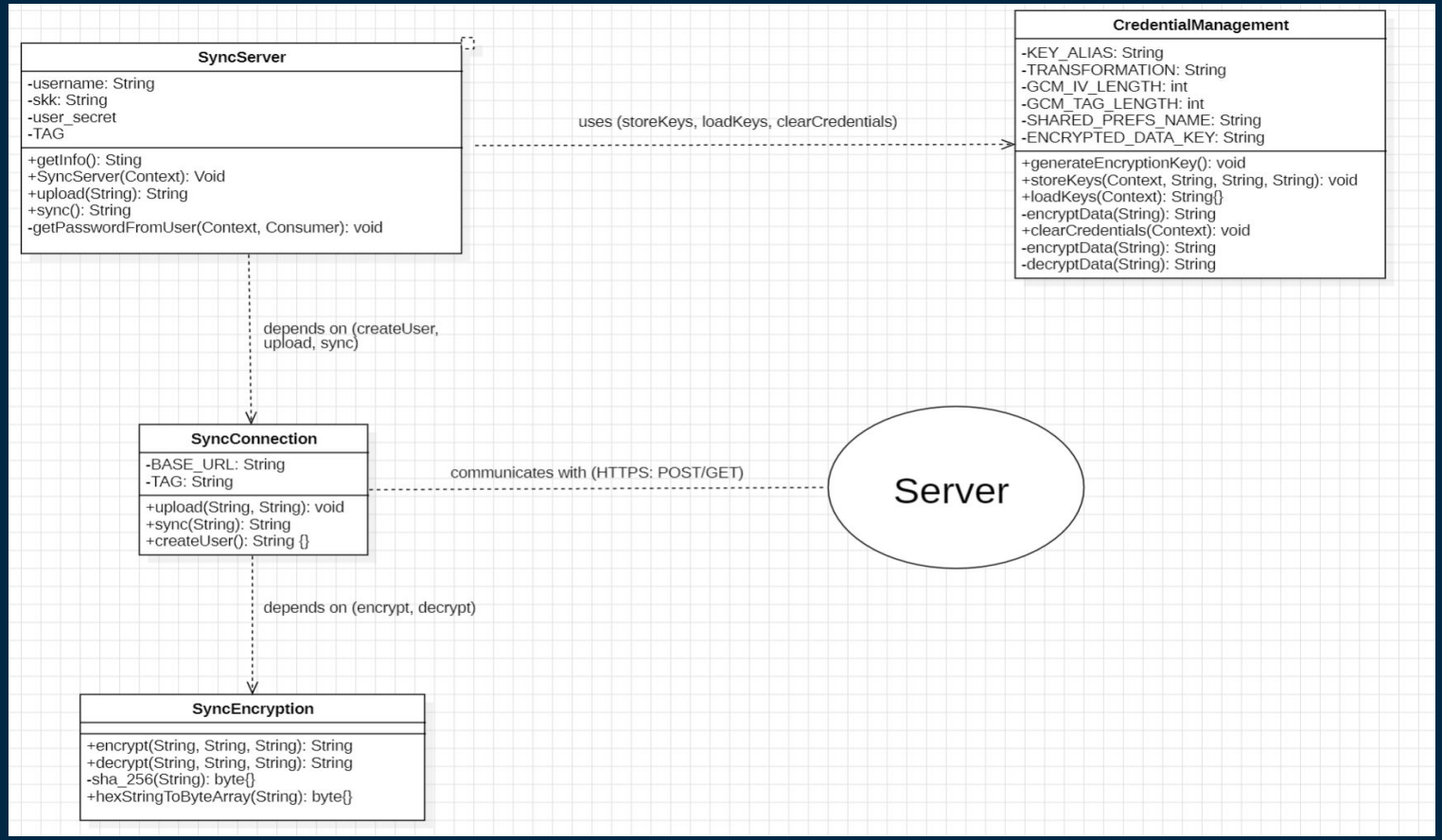
Disk Encryption

- Encrypts sensitive application data before storing it on disk.
- Stores initialization vectors (IVs) and encrypted data securely on the device.
- Keys managed by Android KeyStore, providing strong, hardware-backed encryption.

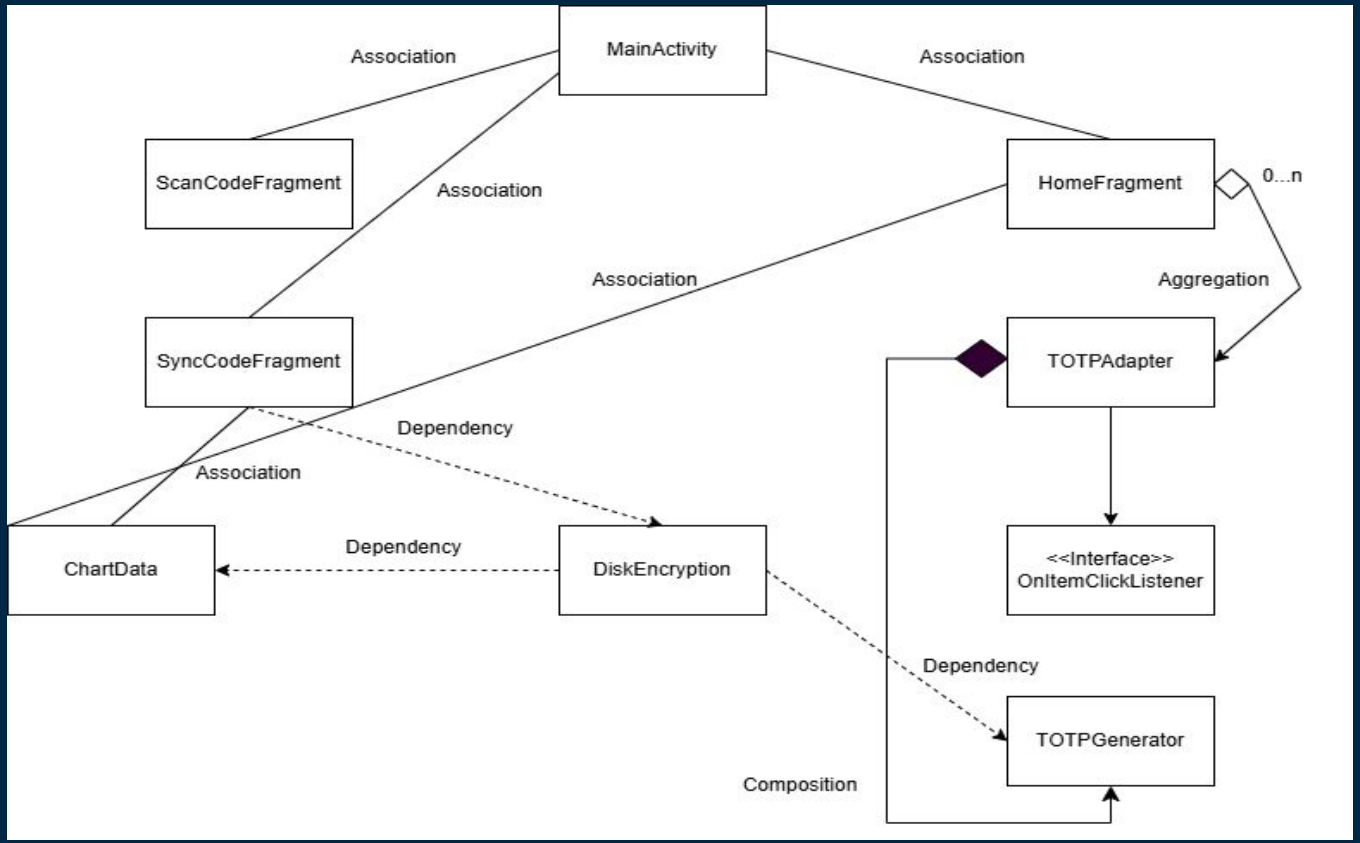
Software design – Server side



Software design – Sync with server



Software design – UI and local storage implementation



Demonstration