System Design Document: SMS-CAS Microservice Architecture with MySQL Database

1. Introduction The purpose of this system design document is to provide a detailed architectural overview and component specifications for a microservice-based system that handles SMS requests through an SMS-CAS interface. The system consists of the following components: SMS-CAS interface, EMMG, Encryptor, Cycler, and Scheduler. The system relies on a MySQL database for data persistence.
2. Architecture Overview The system follows a microservice architecture, where each component is responsible for a specific set of functionalities. The components communicate with each other via RESTful APIs. The SMS-CAS interface receives SMS requests and interacts with the database. EMMG generates EMMs based on SMS data, which are then encrypted by the Encryptor component. The Cycler component carousels encrypted EMMs based on predefined configurations. The Scheduler handles periodic tasks such as key rotation and removal of expired rows from the database.
3. Component Specifications

3.1 SMS-CAS Interface The SMS-CAS interface is responsible for accepting SMS requests and sending REST API requests to various endpoints. It provides the following functionalities:

- Accept SMS requests and extract relevant data.
- Save SMS data in the MySQL database.
- Log requests and responses in a log file.
- Respond to the SMS sender with an appropriate response.

3.2 EMMG (Encrypted Mobile Message Generator) EMMG generates EMMs based on the data received from the SMS-CAS interface. It communicates with the Encryptor component and provides the following functionalities:

- Receive SMS data from the SMS-CAS interface.
- Generate EMMs based on the received data.
- Send EMMs to the Encryptor component for encryption.
- Save the encrypted EMMs in the MySQL database.

3.3 Encryptor The Encryptor component is responsible for encrypting the EMMs received from the EMMG component using envelope encryption. It provides the following functionalities:

- Receive EMMs from the EMMG component.
- Encrypt EMMs using envelope encryption.
- Respond to the EMMG component with the encrypted EMMs.

Requirements for the Encryptor component:

- Support a minimum of 100 encryptions per second.

3.4 Cycler The Cycler component carousels the encrypted EMMs based on a fixed configuration. It provides the following functionalities:

- Retrieve encrypted EMMs from the MySQL database.
- Perform carousel operations on the encrypted EMMs based on predefined configurations.
- Log all data being carouselled.

3.5 Scheduler The Scheduler component handles scheduled tasks and maintenance activities. It provides the following functionalities:

- Run fixed tasks, such as key rotation.
- Remove rows from the MySQL database for which the expiry has passed.

4. Database Design The system utilizes a MySQL database for data persistence. The database design includes the following tables:

4.1 SMS Table

- Columns: id (primary key), sender, message, received_at, processed_at, response
- This table stores the SMS requests and relevant information.

4.2 EMM Table

- Columns: id (primary key), emm_data, encrypted_emm, created_at
- This table stores the generated EMMs and their encrypted versions.

5. Integration and Communication The components communicate with each other using RESTful APIs. Each component exposes specific endpoints for sending and receiving data. The APIs should be secured with appropriate authentication and authorization mechanisms.
6. Deployment and Scalability The microservices should be deployed as independent containers, such as Docker containers, to achieve scalability and maintain separation of concerns. Each component can be deployed on separate servers or orchestrated using container orchestration platforms like Kubernetes for better scalability and fault tolerance.
7. Conclusion The proposed system design outlines a microservice architecture with a MySQL database for handling SMS requests and generating encrypted EMMs. Each component performs its specific functionalities, communicating through RESTful APIs. The design allows for flexibility, scalability, and easy maintenance of the system.