**Record Signing Service Design Document**

**Introduction**

The record signing service is a message-driven/microservice solution that signs batches of records concurrently, storing the signatures in a database until all records are signed. The service is implemented in Golang and uses RabbitMQ as the message broker and MongoDB as the database. The key management service and the record signing service run as separate microservices and communicate with each other via RabbitMQ. The key management service retrieves the least recently used key from the keyring and sends it to the record signing service, which signs the next batch of records with the key and stores the signatures in the MongoDB database. The keyring keeps track of the usage of the private keys to ensure that each key is used only once per batch. The user can configure the batch size for signing records.

**Architecture**

The architecture of the record signing service consists of the following components:

**RabbitMQ**

RabbitMQ is used as the message broker for communication between the key management service and the record signing service. The key management service sends key requests to the record signing service, and the record signing service sends signed records back to the key management service.

**Key Management Service**

The key management service is responsible for managing the private keys used for signing records. It maintains a keyring that keeps track of the usage of the private keys and retrieves the least recently used key from the keyring when a key is needed for signing records. The key management service listens for key requests from the record signing service via RabbitMQ and sends the least recently used key back to the record signing service.

**Record Signing Service**

The record signing service is responsible for signing batches of records and storing the signatures in the MongoDB database. It retrieves unsigned records from the MongoDB database, requests a key from the key management service via RabbitMQ, signs the records with the key, and stores the signed records back in the MongoDB database. The record signing service listens for key responses from the key management service via RabbitMQ.

**Keyring**

The keyring is a data structure that keeps track of the private keys used for signing records and their usage. It maintains a list of keys and a map of key usage times, and provides a method for retrieving the least recently used key.

**MongoDB**

MongoDB is used as the database for storing the records and their signatures. The MongoDB collection is initialized with 100,000 records of random data.

**Implementation**

The record signing service is implemented in Golang and consists of the following functions:

**KeyManagementService**

The KeyManagementService function is responsible for managing the private keys used for signing records. It connects to the RabbitMQ broker, declares two queues for key requests and responses, and consumes messages from the key request queue. When a message is received, it gets the batch size from the message body and retrieves the least recently used key from the keyring. It then sends the key back in a response message and waits for a short time to simulate key signing. Finally, it signs the next batch of records with the key, logs the signing, and repeats the process for the next message.

**SignRecords**

The SignRecords function signs a batch of records with a given private key. It takes a slice of records and a private key as input, and iterates over the records, signing each one with the key and updating its signature.

**Sign**

The Sign function signs a string of data with a given private key. It takes a string of data and a private key as input, decodes the private key from base64, parses it as an RSA private key, calculates the SHA-256 hash of the data, signs the hash using the private key, and encodes the signature as base64 and returns it.

### SaveRecords

The SaveRecords function saves a batch of records to the MongoDB database. It takes a slice of records as input, converts the records to MongoDB documents, and inserts the documents into the MongoDB collection.

### GetUnsignedRecords

The GetUnsignedRecords function retrieves up to a given batch size of unsigned records from the MongoDB database. It takes a batch size as input, finds unsigned records in the MongoDB collection, and returns them as a slice of records.

### NewKeyring

The NewKeyring function initializes a new keyring with a list of keys and an empty map of key usage times. It takes a slice of keys as input and returns a pointer to a new keyring.

### main

The main function initializes the keyring with 100 private keys, initializes the MongoDB collection with 100,000 records of random data, and starts the key management service and the record signing service as separate Goroutines. It takes no input and returns no output.

## Configuration

The record signing service can be configured with the following parameters:

### batchSize

The batchSize parameter specifies the number of records to sign in each batch. It is an integer value and can be set to any positive integer.

## Dependencies

The record signing service has the following dependencies:

### RabbitMQ

RabbitMQ is used as the message broker for communication between the key management service and the record signing service. The RabbitMQ broker must be installed and running on the system.

### MongoDB

MongoDB is used as the database for storing the records and their signatures. The MongoDB server must be installed and running on the system.

## Conclusion

The record signing service is a message-driven/microservice solution that signs batches of records concurrently, storing the signatures in a database until all records are signed. It is implemented in Golang and uses RabbitMQ as the message broker and MongoDB as the database. The key management service and the record signing service run as separate microservices and communicate with each other via RabbitMQ. The keyring keeps track of the usage of the private keys to ensure that each key is used only once per batch. The user can configure the batch size for signing records.

**Out of scope**

There are several improvements that could be made to the record signing service:

1. Use a load balancer to distribute incoming requests to multiple instances of the record signing service for better scalability and fault tolerance.
2. Use a caching layer to cache frequently accessed data in memory for faster access and reduced load on the database.
3. Implement a health check endpoint to monitor the status of the record signing service and alert administrators in case of issues.
4. Implement access control to restrict access to the record signing service and its data to authorized users only.
5. Use a more secure method for storing the private keys, such as a hardware security module (HSM), to prevent unauthorized access to the keys.
6. Use a more secure algorithm for signing the records, such as Ed25519 or ECDSA, which are considered to be more secure than RSA.
7. Implement automatic key rotation to periodically replace the private keys used for signing records to prevent compromise of the keys over time.