Technical Report: Secure RESTful API with Encrypted Data Storage and Key Management

1 System Architecture Overview

The system is a secure RESTful API designed to provide user authentication, secure management of sensitive data, and cryptographic key management. It is developed using the FastAPI framework, chosen for its high performance, support for asynchronous programming, and automatic OpenAPI documentation generation. Data storage is handled by MongoDB, a flexible NoSQL database that supports horizontal scalability and efficient management of structured and semi-structured data. Cryptographic key management is securely isolated using HashiCorp Vault, an industry-standard tool for managing secrets and keys.

The system is deployed via Docker Compose, which facilitates the management and deployment of multiple services in isolated containers, ensuring consistency between development and production environments. The architecture consists of three main components:

- **FastAPI Application**: Responsible for handling HTTP requests, user authentication via JWT tokens, encryption and decryption of sensitive data using user-specific keys, and interaction with MongoDB and HashiCorp Vault.
- **MongoDB Database**: Stores user information (e.g., username, hashed password, salt, and two-factor authentication data) and encrypted sensitive data (including data type and encrypted value).
- **HashiCorp Vault**: Manages cryptographic keys securely, including the master key and individual user keys encrypted with the master key.

This architecture ensures that sensitive data remains encrypted during transmission and storage, with keys managed separately and securely.

2 Cryptographic Algorithms and Hashing

To ensure data security and user authentication, the system employs standard cryptographic and hashing algorithms:

- **Password Hashing**: The bcrypt algorithm with a work factor of 12 is used to hash passwords, providing strong resistance against brute-force attacks. Each user has a unique 16-byte salt generated via `secrets.token_hex(16)` to prevent rainbow table attacks. Additionally, a global pepper stored in the `.env` file is appended

to the password before hashing for extra security. The final hash is stored in the `hashed_password` field.

- **Sensitive Data Encryption**: AES-256-GCM, a symmetric encryption algorithm with authentication, is used. Each user has a unique 256-bit key, encrypted with the master key and stored in Vault. A unique 12-byte nonce is generated for each encryption operation using `secrets.token_bytes(12)` to ensure ciphertext uniqueness.
- **JWT Token Signing**: JWT tokens are signed using the HS256 (HMAC-SHA256) algorithm with a 1-hour expiration. The signing key (`JWT_SECRET`) is securely stored in the `.env` file.

These algorithms were selected for their high security, standardization, and broad support in cryptographic libraries, ensuring that passwords are irretrievable, sensitive data is securely encrypted, and authentication tokens are tamper-proof.

3 Database Structure

The MongoDB database, named `secure_api`, consists of two main collections:

- **`users` Collection**: Contains user information with the following fields:
  - `_id`: Unique identifier (ObjectId).
  - `username`: Unique username.
  - `email`: Email address for two-factor authentication.
  - `hashed_password`: Bcrypt hash of the password with salt and pepper.
  - `salt`: Unique 16-byte salt.
  - `failed_attempts`: Count of failed login attempts (max 5).
  - `last_failed_attempt_time`: Timestamp of the last failed attempt for a 15-minute lockout.
  - `totp_secret`: Base32 secret for TOTP-based two-factor authentication.
  - `two_factor_enabled`: Status of two-factor authentication.
- **`sensitive_data` Collection**: Stores encrypted sensitive data with the following fields:
  - `_id`: Unique identifier (ObjectId).
  - `user_id`: Reference to the user's `_id` in the `users` collection.
  - `data_type`: Type of sensitive data (e.g., "card_number").

  - `encrypted_value`: Hex-encoded encrypted data (including nonce and ciphertext).

This structure ensures the separation of authentication and sensitive data, guaranteeing secure and encrypted storage.

4 Key Management

Key management is handled using HashiCorp Vault as follows:

- **Master Key**: A 256-bit AES key stored at `kv/master_key` in Vault. It is used to encrypt and decrypt user-specific keys and is automatically generated and stored if not present.
- **User Keys**: Each user has a unique 256-bit AES key generated during registration, encrypted with the master key, and stored with a nonce at `kv/user_keys/<username>` in Vault.
- **Key Rotation**: The `/rotate-master-key` endpoint performs key rotation by:
  1. Retrieving the old master key.
  2. Generating a new master key.
  3. Decrypting each user's key with the old master key and re-encrypting it with the new one.
  4. Updating the master key in Vault.

This method ensures key security and allows key rotation without altering sensitive data.

5 Tool and Method Selection

The tools and methods were chosen based on the following criteria:

- **FastAPI**: Selected for its high performance, asynchronous support, automatic OpenAPI documentation, and data validation with Pydantic, making it ideal for secure and scalable API development.
- **MongoDB**: Chosen as a NoSQL database for its flexibility in handling structured and semi-structured data, and seamless integration with FastAPI via the `motor` library.
- **HashiCorp Vault**: Selected for its high security, access control, and key rotation capabilities as an industry-standard tool.

- **Docker Compose**: Used for easy deployment and management of multi-container services, ensuring environment consistency.
- **Cryptography Library**: Employed for its secure and tested implementation of cryptographic algorithms like AES-256-GCM.

This combination of tools and methods provides a secure, scalable, and maintainable system that meets all project requirements.