1. Выбор схемы хранения конфиденциальной информации

Конфиднциальные данные будут храниться в базе данных. Для MongoDB можно организовать хранение способами:

- Encryption at Rest;
- Client-Side Field Level Encryption;
- Manage by yourself.

Encryption at Rest позволяет защитить данные на уровне базы данных. MongoDB будет шифровать файлы данных с помощью выбранных алгоритмов. Это позволит сохранить данные в безопасности, если злоумышленник получит доступ к серверу с базой данных. Но это не спасет от утечки, если злоумышленик получит доступ пользователю базы данных или произведет иньекцию в БД.

Client-Side Field Level Encryption позволяет шифровать данные перед вставкой и расшифровать после чтения автоматически с помощью Encryption Client. При использова обычного клиента данные будут зашифрованы. Данный способ не спасает от утечки при получении доступа к Encryption Client. Так же доступно только два алгоритма шифрования:

- AEAD_AES_256_CBC_HMAC_SHA_512_Random;
- AEAD_AES_256_CBC_HMAC_SHA_512_Deterministic.

Из них можно использовать только один из-за отсутвия соли в другом.

Самостоятельное управление процессом шифрования позволит решить данные проблемы, но потребует больше затрат на разработку. Остается уязвимое место с хранением ключа шифрования, но оно решается использованием Key management system.

2. Выбор алгоритмов и библиотек для хранения конфиденциальной информации

Выбор производился на основе статей Cryptographic Storage Cheat Sheet и Envelope encryption. Согласно им необходимо выбрать алгоритм реализовывающий AEAD и использовать его с длинными ключами, которые необходимо заменять время от времени. А управление ключами производить с помощью KMS, используя концепты Key encryption keys (KEK) и Data encryption keys (DEK).

Для эмуляции работы с KMS использовал Local KMS (моковая реализация AWS KMS) с доступом к нему через aws-sdk.

Для шифрования использовал библеотеку libsodium из предыдущей лабораторной, которая предоставляет AEAD ciphers. Согласно документации она предоставляет алгоритм XChaCha20-Poly1305, который на данный момент может безопасно зашифровать практически неограниченное количество сообщений любого размера, а случайные nonces безопасны в использовании. Параметры алгоритма (Key size, Nonce size, Block size, MAC size), его ограничения не нарушают рекомендаций статей, а сравнительный тесты

производительностей показывают выигрыш перед алгоритмами класса AES, поэтому можно считать данный алгоритм подходящим.

При шифровании конфиденциальной информации для каждой записи используется уникальный DEK, который храниться рядом с данными в зашифрованном виде. Приложение не имеет прямого доступа к KEK и для него включена автоматическая ротация в KMS.

3. Возможные атаки

Уязвимым местом остается хранение Credentials для AWS. При получению доступа к ним у злоумышленников будет возможность расшифровать любой DEK. Поэтому достаточно важно настроить безопасное хранение их с помощью файлов с соотвествующими правами доступа.

Так же несмотря на заявления libsodium по поводу XChaCha20-Poly1305, необходимо проводить переодическое перешифрование, если использовать DEK несколько раз (в случае ограничений связаных с стоимостью использования KMS).