# JetStream Encryption At Rest

| Metadata | Value |
|---|---|
| Date | 2021-07-20 |
| Author | @derekcollison |
| Status | Implemented |
| Tags | jetstream |

## Context and Problem Statement

Present a way to have the nats-server itself provide encryption at rest for all stored JetStream data. This should be seen as an alternative to an encrypted filesystem or block storage that would be provided by a cloud provider.

End user documentation for this feature can be found in the official documentation.

## Design

The design will allow a master key to be used to generate keys for asset encryption within JetStream. The "assets" are *account* information, and metadata and data for *streams* and *consumers*. Even within stream data that can be broken into multiple message blocks on disk, each block will have a different asset key that will be used to encrypt.

## Keys

- *EK* - External Key, provided to the nats-server configuration or via encryption service in subsequent releases.
- *KEK* - An encryption key derived from the *EK*. Always generated and never stored. This is generated via a PRF where `KEK := PRF(EK, IV, Context)`, e.g. `HMAC-SHA-256`, `Argon2Id`
- *AEK* - Asset encryption key, encrypted via a *KEK* and placed alongside the asset on disk.

## Encryption

Encryption will utilize an authenticated encryption AEAD. We use ChaCha20-Poly1305.

## Future Support

### KMS

In a follow-on release we may choose to configure the nats-servers to access a KMS and not have direct access to the *EK*. Today the recommended method is to use an environment variable as demonstrated in the documentation.

### Key Rolling

In the initial release we will have different keys for each message block for a stream, meaning the *KEK* and *AEK* will not be re-used for any subsequent message blocks.

However, we may want to introduce the ability to roll keys which could be TBD. For the first release a backup and restore will be equivalent to changing all of the encryption keys used to encrypt the stream asset.

We would want to think through rotating the *EK* as well at some point. We would need access to the old one for retrieving older assets.

## Client Side EK

We could also consider allowing clients to provide this *EK* in a header such that even if the server had a KMS the client could control this on their own. Of course this requires TLS (which we normally promote) and is not as good as a KMS or a CP provided solution like encrypted EBS etc.