KIP-772: Encrypt KRaft Metadata Secrets at Rest

Status

Current state: *"Under Discussion"*

Discussion thread:

JIRA:

Motivation

Precedent

The KIP which established dynamic broker configuration, KIP-226, specified that this configuration data would be stored in ZooKeeper. It also established an encryption mechanism for secrets such as passwords. We would like to have the same level of protection for secret data in the post-ZooKeeper world of KRaft.

Threat Model

The threat model for the KIP-226 encryption was to protect against an attacker who had compromised the ZooKeeper service, but not the Kafka service. Since there is no more ZooKeeper service when using KRaft, our model is different. Our threat model is to protect against an attacker who has gained access to the metadata log, but not other aspects of the system (such as the system used to store encryption keys).

Different users may wish to store metadata encryption keys in different ways. For example, in a system with both local storage and remote network-accessed storage, it may make sense to store the metadata encryption keys on the local storage, but the metadata log on remote-accessed storage. Alternately, some users may want to store the metadata encryption key remotely, so that it doesn't appear on disk storage accessible to the broker at any time. A third user may not want to encrypt metadata at all. Therefore, this KIP makes the metadata encryption key mechanism both pluggable and optional.

Key Rotation

Finally, metadata key rotation is very important to security. We want it to be possible to do this rotation online, without shutting down the cluster and taking downtime. It should also be possible to do this without doing a rolling restart.

Overview

Scope of Encryption

As mentioned previously, only secrets in the metadata log will be encrypted by this mechanism. That includes things like configurations which are passwords, and new metadata encryption keys themselves.

Metadata Encryption Keys

Overview

Metadata encryption keys are sequences of bytes used to encrypt the metadata via a symmetrical encryption algorithm. Each key is identified by a 16-byte UUID.

Registration

Metadata keys may be "registered," which associates them with a MetadataEncryptionKeyAccessor class.

Populated

A registered metadata key is "populated" when the key data is present.

Activated

A registered and populated metadata key may be "activated," which means that it becomes the key which is actually used for metadata encryption. Only one key can be active at once.

In order to facilitate key rotation, many metadata encryption keys may be "registered" with the system. However, only one key can be "active." The active key is the one which is actually used for encryption.

Each metadata encryption key is identified by an unique 16-byte UUID.

```
Registered
    |
    |
    V
```

Mechanism

In general,

Public Interfaces

MetadataEncryptionKeyAccessor

RegisterMetadataEncryptionKeyRecord

This record registers a metadata encryption key in the metadata log. This record appears once in the metadata log for each registered encryption key. It also appears once in the metadata image for each currently registered metadata encryption key.

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "MetadataEncryptionKeyRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "KeyId", "type": "uuid", "versions": "0+",
      "about": "The metadata encryption key ID." },
    { "name": "AccessorClass", "type": "string", "versions": "0+",
      "about": "The MetadataEncryptionKeyAccessor class to use for this key." }
  ]
}
```

TransmitMetadataEncryptionKeyRecord

This record transmits a metadata encryption key in the metadata log. This record appears once in the metadata log for each registered encryption key that needs to be transmitted. It also appears once in the metadata image for each currently registered metadata encryption key.

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "MetadataEncryptionKeyRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "KeyId", "type": "uuid", "versions": "0+",
      "about": "The metadata encryption key ID." },
    { "name": "AccessorClass", "type": "string", "versions": "0+",
      "about": "The MetadataEncryptionKeyAccessor class to use for this key." }
  ]
}
```

ActivateMetadataEncryptionKeyRecord

MetadataEncryptionKeyReferenceRecord

This record appears in the metadata image

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "RegisterMetadataEncryptionKeyRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "KeyId", "type": "uuid", "versions": "0+",
      "about": "The metadata encryption key ID." },
    { "name": "AccessorClass", "type": "string", "versions": "0+",
      "about": "The MetadataEncryptionKeyAccessor class to use for this key." }
  ]
}
```

RegisterMetadataEncryptionKeyRecord

This record registers a metadata encryption key, which is identified by a 16-byte UUID. The "mechanism" field is the full name of a Java class which implements the given metadata encryption key. Note that this record does not contain the actual encryption key. It simply registers it.

This record appears once in the metadata log and once in the metadata snapshot for each registered encryption key.
nregisterMetadataEncryptionKeyRecord

This record unregisters a metadata encryption key.

This record appears once in the metadata log for each registered encryption key that needs to be deregistered. It does not appear in the metadata snapshot.

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "UnregisterMetadataEncryptionKeyRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "KeyId", "type": "uuid", "versions": "0+",
      "about": "The metadata encryption key ID." }
  ]
}
```

ctivateMetadataEncryptionKeyRecord

This record activates a metadata encryption key in the metadata log.

This record appears once in the metadata log each time we activate a new metadata encryption key. It appears once at the beginning of each metadata snapshot.

```
{
  "apiKey": ...,
  "type": "metadata",
  "name": "ActivateMetadataEncryptionKeyRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "EncryptorId", "type": "uuid", "versions": "0+",
      "about": "The ID of the encryptor to install, or all zeroes to install no encryptor." }
  ]
}
```

create encryptor record

delete encryptor record

encrypted config record

default encryptor

(how to add new encryptor)

Compatibility, Deprecation, and Migration Plan

Rejected Alternatives