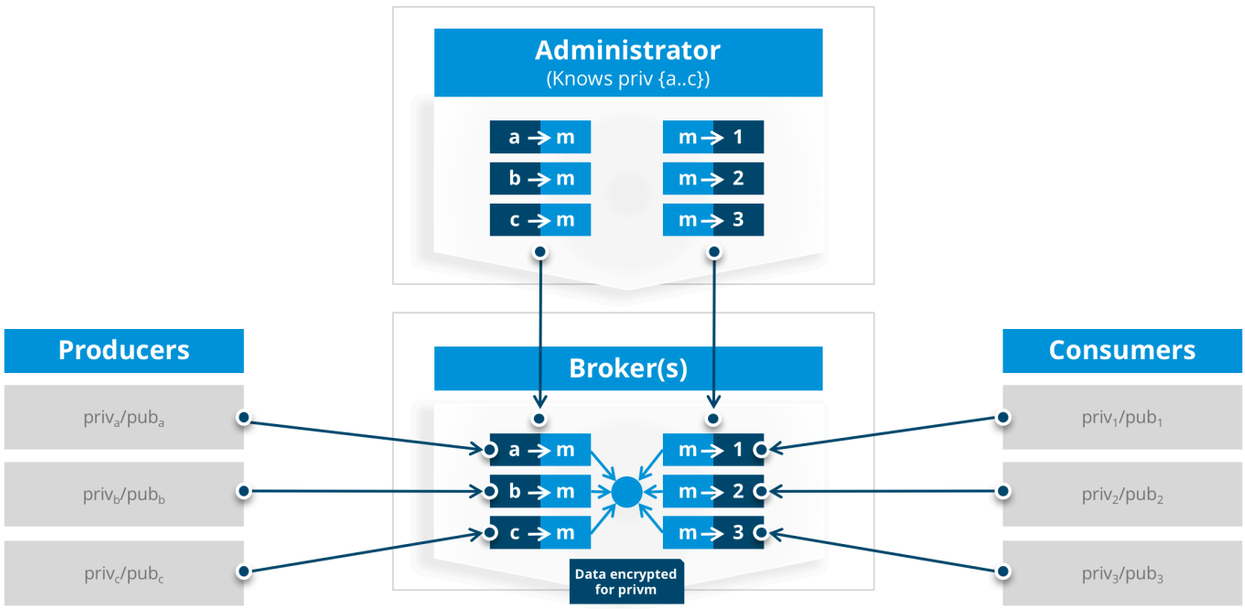
NUCYPHER

NuCypher Kafka: Delegated Access Control and Encryption Management System

HIGH-PERFORMANCE ACCESS MANAGEMENT AND DATA PROTECTION FOR KAFKA MESSAGE STREAMS

Here we describe a delegated access scheme for encrypted message queues and streaming. For simplicity, we assume one broker and one channel. It is trivial to expand to multiple brokers and channels.

The scheme relies on proxy re-encryption which is a method of transforming data encrypted under one key to another key without an intermediate decryption step.



Public/private key pairs denoted here are the following:

- priv$_m$/pub$_m$: key pair under which data is encrypted most of the time on the broker side. If messages are stored for a prolonged time, they are encrypted under this key pair;
- priv$_a$/pub$_a$ .. priv$_c$/pub$_c$: key pairs under which producers encrypt the data. The producers don't have to know current value of pub$_m$;
- priv$_1$/pub$_1$ .. priv$_3$/pub$_3$: key pairs of consumers. Their own private keys can decrypt data they're receiving;
- Administrator knows all private keys of producers, priv$_m$, and private keys of consumers (or just public key of consumers, depending on the variant of proxy re-encryption used).

When a producer *a* connects to the broker, it generates a random AES key per session (DEK$_a$). It includes an encrypted version of it, EDEK$_a$ = enc(pub$_a$, DEK$_a$), as a part of every message. The content of the message is encrypted with DEK$_a$ while the topic is public.

On the broker side, there are re-encryption keys k_{xm} to transform data from key *x* to key *m*. The system administrator responsible for granting permissions knows all the private keys and generates the re-encryption keys.

Each consumer has an individual public/private key pair, let's take $priv_1/pub_1$ as an example. The broker layer holds a re-encryption key k_{m1}, so it transforms $EDEK_m$ -> $EDEK_1$ to be readable by consumer 1 if $EDEK_m$ wasn't yet re-encrypted for 1. If that EDEK was already re-encrypted for consumer 1, the cached version is used.

After that, the consumer can decrypt $EDEK_1$ with his own key $priv_1$. For performance, cached DEK_a can be used if it was already decrypted for this $EDEK_1$. The bulk of the message data, encrypted with DEK_a, can be decrypted with DEK_a.

We leave out the details of message/producer authentication because these questions are solved elsewhere. But it's worth noting that if producers produce message signatures, the broker can convert them to "per channel" signatures if desired.