KIP-651 - Support PEM format for SSL certificates and private key

Status

Current state: *Accepted*

Discussion thread: https://lists.apache.org/thread.html/r4e48ab3433c2a7d52c341dd309c1b3016b03fb82c7f2af99463f4166%40%3Cdev.kafka.apache.org%3E

JIRA: ⚠ Unable to render Jira issues macro, execution error.

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

Kafka currently supports only file-based key and trust stores for SSL. Both broker listeners and clients are configured using these settings:

- `ssl.keystore.type` (JKS or PKCS12)
- `ssl.keystore.location`
- `ssl.keystore.password`
- `ssl.key.password`
- `ssl.truststore.type (JKS or PKCS12)`
- `ssl.truststore.location`
- `ssl.truststore.password`

KIP-226 added support for dynamic reconfiguration of SSL key stores and trust stores. This KIP also enabled password configs to be stored encrypted in ZooKeeper. KIP-297 and KIP-421 added support for externalizing configs, enabling sensitive configs to be stored in a secure third party store. The use of file-based configs makes management of SSL stores and associated passwords difficult. For example, when broker key stores are updated prior to expiration, key stores need to be first distributed to the broker machines. This cannot be done using the Kafka protocol. After distributing the files, Kafka API is used to update the stores in brokers. Brokers load the key/trust material from the file system and the associated password from broker configs, which may be stored in ZooKeeper or an external vault for secret protection.

Privacy-Enhanced Mail (PEM) is a standard format for storing and distributing cryptographic keys, certificates and other data, defined in RFC-1421 , RFC-1422, RFC-1423 and RFC-1424. Though no longer used as the standard for electronic mail, the container format defined by PEM is widely used for storing cryptographic keys and certificates. PEM is supported by OpenSSL and is used by Netty as the standard for certificates and keys its APIs. The SSL providers in Kafka have been made customizable using new security providers introduced in KIP-492 and custom SSL engine factory introduced in KIP-519 . Custom implementations can use third party libraries to load key and trust stores from different sources in different formats including PEM. But it will be good to standardize configs that can be used with inline PEM for keys and certificates without relying on files.

Goals:

- Add support for PEM files in addition to existing JKS/PKCS12 for key and trust stores. This enables use of third party providers that use PEM.
- Add new configurations to provide private key and certificates directly in PEM format without relying on files. This avoids the need to maintain and protect both Kafka config files and separate key store files.
- Support dynamic config updates of SSL private keys and certificates using Kafka protocol, without relying on a side channel for propagation of files.
- Support secret protection for SSL private keys through externalization or encryption, without also requiring to protect files on the file system.
- Protect PEM data using encryption when configured as dynamic configs, stored in ZooKeeper.
- Improve detection of certificate or private key change. We currently check file modification times since it is diffcult to determine if certs in JKS /PKCS12 have changed. We can do String comparison of PEM files instead.
- Avoid dependency on third party libraries in the default implementation.

- Include limited support for encrypted private keys in PEM format using standard Java libraries. Third party libraries like `bouncycastle` can be used to add custom `SslEngineFactory` implementation to support wider range of options for loading encrypted keys.

Public Interfaces

The following new configuration options will be introduced by this KIP:

ssl.keystore.key

- Type: `Password`
- Description: Private key in the format specified by `ssl.keystore.type`. Default SSL engine factory will support only PEM format with PKCS#8 keys. If the key is encrypted, key password must be specified using ``ssl.key.password``.

ssl.keystore.certificate.chain

- Type: `Password`
- Description: Certificate chain in the format specified by `ssl.keystore.type`. Default SSL engine factory will support only PEM format with a list of X.509 certificates.

ssl.truststore.certificates

- Type: `Password`
- Description: Trusted certificates in the format specified by `ssl.truststore.type`. Default SSL engine factory will support only PEM format with X.509 certificates.

Even though public certificates don't need to be hidden, `Password` type will be used to be consistent with existing key and trust store configs. Private keys may be provided encrypted for additional security. When unencrypted PEMs are used, dynamic config support or secret protection may be used to secure private keys.

A new key store type ``PEM`` will be added for key and trust stores. Both `ssl.keystore.type` and `ssl.truststore.type` may specify PEM in addition to JKS and PKCS12.

Certificates and private keys may be configured using:

1. existing configs `ssl.keystore.location/ssl.truststore.location` to load PEM files
2. new configs `ssl.keystore.key`, `ssl.keystore.certificate.chain, ssl.truststore.certificates` to specify PEM in server.properties
3. new configs `ssl.keystore.key`, `ssl.keystore.certificate.chain, ssl.truststore.certificates` to store PEM encrypted in ZooKeeper using Admin API or `kafka-configs.sh`
4. new configs `ssl.keystore.key`, `ssl.keystore.certificate.chain, ssl.truststore.certificates` to store PEM in a secure external vault using config providers

Proposed Changes

`DefaultSslEngineFactory` will be extended to load PEM data and create in-memory PKCS12 key stores and trust stores using standard Java APIs. The rest of the implementation will be the same as the existing SSL implementation. To avoid dependency on external libraries, built-in Kafka provider will only support limited forms of password-encryption for PEM files. Third-party libraries like `bouncycastle` may be used to extend the SSL engine builder to support a wider range of encryption and storage formats.

PEM files are multi-line configs and may be configured on multiple lines. The values will be trimmed to generate the Base64 encoded string. When updating certificates using `kafka-configs.sh`, the ``--add-config-file`` option can be used to simplify updates on the command line.

For example:

```
ssl.keystore.certificate.chain=-----BEGIN CERTIFICATE----- \
MIIC4jCCAcqgAwIBAgIIJHw42Lu1+w8wDQYJKoZIhvcNAQEFBQAwJDEPMA0GA1UE \
AwwGY2xpZW50MREwDwYDVQQKDAhBIGNsaWVudDAeFw0yMDA4MDMwOTU4MTZaFw0y \
MDA5MDIwOTU4MTZaMCQxDzANBgNVBAMMBmNsaWVudDERMA8GA1UECgwIQSBjbGll \
bnQwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIBAQCwTUf499MROpsz8LFr \
EOZEvUH6e1qks6AJEWjD7BY/SmwRijNPAaJhHaogYaVPrDEmFfexZDVhtc4eDkDI \
rW6+ZlkpNZupkINCR4R49f7JCjFz6rwGl4lSpa3mIhkXS/ZD0pjCYB9t2xBuTWVq \
ap40WqbQDsJHNH+9V/nzktX0ZOB6AgUuzFwLu3YDKS8XFD5TAdZKIu8rtxFzL1Uo \
HmiWFU9EoHROs23xJn7jCEOBq3L2b5IEE/ZHZVw/ooi/jJIID21bkiI731RWOoE3 \
ClEsh7CQHWlXwyoJmMP2dZrXbERpZclH0ozb5JJwJiMtB1uxUiD3wKF/rlcfRAcZ \
AR4vAgMBAAGjGDAWMBQGA1UdEQQNMAuCCWxvY2FsaG9zdDANBgkqhkiG9w0BAQUF \
AAOCAQEAOqNAWknyUljdFeC/O5fDwoGYqHJY3dkinhjfiDEQm+RLLli64xjlNyRJ \
u4ZMHqEE4yQBnQGFxHkKIcA/poDgntSJrSFsfnpHzZJ5kz5zQdNDT9BYQIPWqoe2 \
0plNB6NjZeUn2OH+hAJIbclye0PXMrLwnDVUOJPS9xnlfgbrvIM0HCjtG95oeWv4 \
VLLOKaxiNYEX0xx9fT/lKjnqgi7OPAMTvfp5y1t4BCoe/43o8Pd0Ih2hdgVE6rLn \
mxEaTdlbQNp1ju70Ztl3NNt17+tceq0VbfTRI1xufTB5dCPWeeg0ekC9jMMs42R+ \
PiGYp7h8A3hRC5m8pYnKLSJp5ymITg== \
-----END CERTIFICATE-----
```

```
ssl.keystore.key=-----BEGIN ENCRYPTED PRIVATE KEY----- \
... \
-----END ENCRYPTED PRIVATE KEY-----
```

Compatibility, Deprecation, and Migration Plan

We will continue to support file-based JKS and PKCS12 key and trust stores for SSL. The new configs will only take effect if explicitly set and we will throw InvalidConfigurationException if both file and PEM values are configured at the same time.

Rejected Alternatives

Use custom factory instead of built-in implementation for PEM support

Since custom SSL engine factories can already load certs using external libraries, we can leave it to individual users to add support for PEM if needed. But we believe this is not an uncommon case and having a configuration mechanism that integrates well with other features like secret protection may be useful to many users. Hence a standard set of configs to support this feature seems useful.