KIP-492: Add java security providers in Kafka Security config

Status

Current state: Accepted

Discussion thread: https://www.mail-archive.com/dev@kafka.apache.org/msg99419.html

Voting thread: https://www.mail-archive.com/dev@kafka.apache.org/msg99826.html

JIRA: KAFKA-8669

Motivation

Currently kafka supports ssl.keymanager.algorithm and ssl.trustmanager.algorithm parameters as part of secure config. These parameters can be configured to load the key manager and trust managers which provide keys and certificates for ssl handshakes with the clients/server. The algorithms configured by these parameters need to be registered by Java security provider classes. These provider classes are configured as JVM properties through java.security file. A sample file given below

```
$ cat /usr/lib/jvm/jdk-8-oracle-x64/jre/lib/security/java.security
...
security.provider.1=sun.security.provider.Sun
security.provider.2=sun.security.rsa.SunRsaSign
security.provider.3=sun.security.ec.SunEC
security.provider.4=com.sun.net.ssl.internal.ssl.Provider
security.provider.5=com.sun.crypto.provider.SunJCE
security.provider.6=sun.security.jgss.SunProvider
security.provider.7=com.sun.security.sasl.Provider
security.provider.8=org.jcp.xml.dsig.internal.dom.XMLDSigRI
security.provider.9=sun.security.smartcardio.SunPCSC
...
```

Custom keymanager and trustmanager algorithms can be used to supply the kafka brokers with keys and certificates, these algorithms can be used to replace the traditional, non-scalable static keystore and truststore jks files.

To take advantage of these custom algorithms, we want to support java security provider parameter in security config. For instance, in streaming applications like Flink, Spark or Storm, one can configure 'spiffe.provider.SpiffeProvider' which helps in fetching the keys and certificates via a gRPC endpoint. This param can be used by kafka brokers or kafka clients(when connecting to the kafka brokers). The security providers can also be used for configuring security algorithms in SASL based communication.

Public Interfaces

SecurityProviderCreator interface will be added org.apache.kafka.common.security.auth

```
/**
 * An interface for generating security providers.
 */
@InterfaceStability.Evolving
public interface SecurityProviderCreator extends Configurable {

    /**
     * Configure method is used to configure the generator to create the Security Provider
     * @param config configuration parameters for initialising security provider
     */
    default void configure(Map<String, ?> config) {

    }

    /**
     * Generate the security provider configured
     */
    Provider getProvider();
}
```

```
security.provider.classes=com.security.ProviderClass,com.security.ProviderClass2
```

The following line will be added to the kafka server.properties and the configured provider classes will be registered at the time of kafka broker start up

Proposed Changes

We add new config parameter in KafkaConfig named “security.provider.class.names”. The value of “security.provider.class.names” is expected to be a comma separated string representing the providers' full classname. This provider classes will be added to the JVM through Security.addProvider api. Security class can be used to programmatically add the provider classes to the JVM. The provider classes will be added at the starting position in the list of providers and in the order they are passed. So any algorithm implemented in the earlier classes will replace algorithm implementations with the same name in following providers. For SASL, we add the providers after the login modules are configured so that the algorithms in the providers passed can take precedence over the passed SASL login modules.

```
for (String provider : givenProvidersListReversed) {
   // Adds at position 0
   Security.addProvider((Provider) Class.forName(provider).newInstance());
}
```

Compatibility, Deprecation, and Migration Plan

None

Rejected Alternatives

There are few other ways of adding a custom provider

1. Add a new security provider class in the java.security file

A new configuration like the one below can be added to the java.security file

```
security.provider.10=com.security.CustomProvider
```

2. Pass the security policies as system level arguments

```
# Append or override parts of the default java.security file
java -Djava.security.properties=/some/path/my-augmented-security.properties
```

Asking all the clients to do this is harder to achieve similar to the alternative 1

Existing Properties

There already exists a property “ssl.provider”, however there is a problem with using this parameter. The expected value of this property is the name of the algorithm and the provider class registering the algorithm needs to be added as part of static java.security file or system level variables as per the SslContext implementation. Therefore, this param can’t be used for registering a security provider.