Encryption with SSL

SSL Overview

SSL can be configured for encryption or authentication. You may configure just SSL encryption and independently choose a separate mechanism for client authentication, e.g. SSL (authentication_ssl.html#kafka-ssl-authentication), SASL (authentication_sasl/index.html#kafka-sasl-auth), etc. This section focuses on SSL encryption.

By default, Apache Kafka communicates in `PLAINTEXT`, which means that all data is sent in the clear. To encrypt communication, it is recommended to configure all the Confluent Platform components in your deployment to use SSL encryption.

Quick note on terminology: Secure Sockets Layer (SSL) is the predecessor of Transport Layer Security (TLS), and SSL has been deprecated since June 2015. However, for historical reasons, Kafka (like Java) uses the term SSL instead of TLS in configuration and code, which can be a bit confusing. This document uses only the term SSL.

SSL can be configured for encryption or authentication. You may configure just SSL encryption (by default SSL encryption includes certificate authentication of the server) and independently choose a separate mechanism for client authentication, e.g. SSL (authentication_ssl.html#kafka-ssl-authentication), SASL (authentication_sasl/index.html#kafka-sasl-auth), etc. Note that SSL encryption, technically speaking, already enables 1-way authentication in which the client authenticates the server certificate. So when referring to SSL authentication, it is really referring to 2-way authentication in which the broker also authenticates the client certificate.

Note that enabling SSL may have a performance impact due to encryption overhead.

SSL uses private-key/certificates pairs which are used during the SSL handshake process.

- each broker needs its own private-key/certificate pair, and the client uses the certificate to authenticate the broker

- each logical client needs a private-key/certificate pair if client authentication is enabled, and the broker uses the certificate to authenticate the client

Each broker and logical client can be configured with a truststore, which is used to determine which certificates (broker or logical client identities) to trust (authenticate). The truststore can be configured in many ways, consider the following two examples:

1. the truststore contains one or many certificates: the broker or logical client will trust any certificate listed in the truststore
2. the truststore contains a Certificate Authority (CA): the broker or logical client will trust any certificate that was signed by the CA in the truststore

Using the CA (2) is more convenient, because adding a new broker or client doesn't require a change to the truststore. The CA case (2) is outlined in this diagram (https://github.com/confluentinc/confluent-platform-security-tools/blob/master/single-trust-store-diagram.pdf).

However, with the CA case (2), Kafka does not conveniently support blocking authentication for individual brokers or clients that were previously trusted via this mechanism (certificate revocation is typically done via Certificate Revocation Lists or the Online Certificate Status Protocol), so one would have to rely on authorization to block access. In contrast, with case (1), blocking authentication would be achieved by removing the broker or client's certificate from the truststore.

Creating SSL Keys and Certificates

Refer to the Security Tutorial which describes how to create SSL keys and certificates (../tutorials/security_tutorial.html#generating-keys-certs).

Brokers

Configure all brokers in the Kafka cluster to accept secure connections from clients. Any configuration changes made to the broker will require a rolling restart (post-deployment.html#rolling-restart).

Enable security for Kafka brokers as described in the section below. Additionally, if you are using Confluent Control Center or Auto Data Balancer, configure your brokers for:

- Confluent Metrics Reporter

1. Configure the password, truststore, and keystore in the `server.properties` file of every broker. Since this stores passwords directly in the broker configuration file, it is important to restrict access to these files via file system permissions.

```
ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
ssl.truststore.password=test1234
ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
ssl.keystore.password=test1234
ssl.key.password=test1234
```

2. If you want to enable SSL for inter-broker communication, add the following to the broker properties file (it defaults to `PLAINTEXT`):

```
security.inter.broker.protocol=SSL
```

3. Tell the Kafka brokers on which ports to listen for client and inter-broker `SSL` connections. You must configure `listeners`, and optionally `advertised.listeners` if the value is different from `listeners`.

```
listeners=SSL://kafka1:9093
advertised.listeners=SSL://0.0.0.0:9093
```

4. Configure both `SSL` ports and `PLAINTEXT` ports if:

   - SSL is not enabled for inter-broker communication
   - Some clients connecting to the cluster do not use SSL

```
listeners=PLAINTEXT://kafka1:9092,SSL://kafka1:9093
advertised.listeners=PLAINTEXT://0.0.0.0:9092,SSL://0.0.0.0:9093
```

Note that `advertised.host.name` and `advertised.port` configure a single `PLAINTEXT` port and are *incompatible* with secure protocols. Please use `advertised.listeners` instead.

Optional settings

Here are some optional settings:

`ssl.cipher.suites` A cipher suite is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol

- Type: list
- Default: null (by default, all supported cipher suites are enabled)
- Importance: medium

`ssl.enabled.protocols` The list of protocols enabled for SSL connections

- Type: list
- Default: TLSv1.2,TLSv1.1,TLSv1
- Importance: medium

`ssl.truststore.type` The file format of the truststore file.

- Type: string
- Default: JKS
- Importance: medium

Due to import regulations in some countries, the Oracle implementation limits the strength of cryptographic algorithms available by default. If stronger algorithms are needed (for example, AES with 256-bit keys), the JCE Unlimited Strength Jurisdiction Policy Files (http://www.oracle.com/technetwork/java/javase/downloads/index.html) must be obtained and installed in the JDK/JRE. See the JCA Providers Documentation (https://docs.oracle.com/javase/8/docs/technotes/guides/security/SunProviders.html) for more information.

Clients

The new Producer and Consumer clients support security for Kafka versions 0.9.0 and higher.

If you are using the Kafka Streams API, you can read on how to configure equivalent SSL (https://docs.confluent.io/current/clients/javadocs/org/apache/kafka/common/config/SslConfigs.html) and SASL (https://docs.confluent.io/current/clients/javadocs/org/apache/kafka/common/config/SaslConfigs.html) parameters.

If client authentication is not required by the broker, the following is a minimal configuration example that you can store in a client properties file `client-ssl.properties`. Since this stores passwords directly in the client configuration file, it is important to restrict access to these files via file system permissions.

```
bootstrap.servers=kafka1:9093
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password=test1234
```

If client authentication via SSL is required, the client must provide the keystore as well. Please read the additional configurations required in SSL Authentication (authentication_ssl.html#kafka-ssl-authentication).

Examples using `kafka-console-producer` and `kafka-console-consumer`, passing in the `client-ssl.properties` file with the properties defined above:

```
bin/kafka-console-producer --broker-list kafka1:9093 --topic test --producer.config client-ssl.properties
bin/kafka-console-consumer --bootstrap-server kafka1:9093 --topic test --consumer.config client-ssl.properties --from-beginning
```

Optional settings

Here are some optional settings:

`ssl.provider` The name of the security provider used for SSL connections. Default value is the default security provider of the JVM.

- Type: string
- Default: null
- Importance: medium

`ssl.cipher.suites` A cipher suite is a named combination of authentication, encryption, MAC and key exchange algorithm used to negotiate the security settings for a network connection using TLS or SSL network protocol

- Type: list
- Default: null (by default, all supported cipher suites are enabled)
- Importance: medium

`ssl.enabled.protocols` The list of protocols enabled for SSL connections

- Type: list
- Default: TLSv1.2,TLSv1.1,TLSv1
- Importance: medium

`ssl.truststore.type` The file format of the truststore file.

- Type: string
- Default: JKS
- Importance: medium

ZooKeeper

The version of ZooKeeper that is bundled with Apache Kafka does not support SSL.

Kafka Connect

This section describes how to enable security for Kafka Connect. Securing Kafka Connect requires that you configure security for:

1. Kafka Connect workers: part of the Kafka Connect API, a worker is really just an advanced client, underneath the covers
2. Kafka Connect connectors: connectors may have embedded producers or consumers, so you must override the default configurations for Connect producers used with source connectors and Connect consumers used with sink connectors
3. Kafka Connect REST: Kafka Connect exposes a REST API that can be configured to use SSL using additional properties

Configure security for Kafka Connect as described in the section below. Additionally, if you are using Confluent Control Center streams monitoring for Kafka Connect, configure security for:

- Confluent Monitoring Interceptors
- Confluent Metrics Reporter

Configure the top-level settings in the Connect workers to use SSL by adding these properties in `connect-distributed.properties`. These top-level settings are used by the Connect worker for group coordination and to read and write to the internal topics which are used to track the cluster's state (e.g. configs and offsets).

```
bootstrap.servers=kafka1:9093
security.protocol=SSL
ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
ssl.truststore.password=test1234
```

Connect workers manage the producers used by source connectors and the consumers used by sink connectors. So, for the connectors to leverage security, you also have to override the default producer/consumer configuration that the worker uses. Depending on whether the connector is a source or sink connector:

- For source connectors: configure the same properties adding the `producer` prefix.

```
producer.bootstrap.servers=kafka1:9093
producer.security.protocol=SSL
producer.ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
producer.ssl.truststore.password=test1234
```

- For sink connectors: configure the same properties adding the `consumer` prefix.

```
consumer.bootstrap.servers=kafka1:9093
consumer.security.protocol=SSL
consumer.ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
consumer.ssl.truststore.password=test1234
```

- For SSL with the REST API, configure the following additional properties:

Property	Note
`listeners`	List of REST listeners in the format `protocol://host:port,protocol2://host2:port`, where the protocol is either `http` or `https`. When this property is defined, the properties `rest.host.name` and `rest.port` will be ignored.

Property	Note
`rest.advertised.listener`	Configures the listener used for communication between workers. Valid values are either `http` or `https`. If the `listeners` property is not defined or if it contains an `http` listener, the default value for this field is `http`. When the `listeners` property is defined and contains only `https` listeners, the default value is `https`.
`ssl.client.auth`	Valid values are `none`, `requested`, and `required`. It controls whether: 1. the client is required to do SSL/TLS client authentication (`required`) 2. it can decide to skip the SSL/TLS client authentication (`requested`) 3. the SSL/TLS client authentication will be disabled (`none`)
`listeners.https.ssl.*`	You can use the `listeners.https.` prefix with an SSL configuration parameter to override the default SSL configuration which is shared with the connections to the Kafka broker. If at least one parameter with this prefix exists, the implementation uses only the SSL parameters with this prefix and ignores all SSL parameters without this prefix. In no parameter with prefix `listeners.https.` exists, the parameters without a prefix are used.

Note that if the `listeners.https.ssl.*` properties are not defined then the `ssl.*` properties will be used. For a list of all `ssl.*` properties, see Configuration Options (../kafka-rest/docs/config.html#kafkarest-config).

Here is an example that sets the `ssl.*` properties to use SSL connections to the broker, and since `listeners` includes `https` these same settings are used to configure Connect's SSL endpoint:

```
listeners=https://myhost:8443
rest.advertised.listener=https
rest.advertised.host.name=0.0.0.0
rest.advertised.host.port=8083
ssl.client.auth=requested
ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
ssl.truststore.password=test1234
ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
ssl.keystore.password=test1234
ssl.key.password=test1234
```

To configure Connect's SSL endpoint differently than the SSL connections to the broker, simply define the `listeners.https.ssl.*` properties with the correct settings. Note that as soon as any `listeners.https.ssl.*` properties are specified, none of the top level `ssl.*` properties will apply, so be sure to define all of the necessary `listeners.https.ssl.*` properties:

```
listeners=https://myhost:8443
rest.advertised.listener=https
rest.advertised.host.name=0.0.0.0
rest.advertised.host.port=8083
listeners.https.ssl.client.auth=requested
listeners.https.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
listeners.https.ssl.truststore.password=test1234
listeners.https.ssl.keystore.location=/var/private/ssl/kafka.server.keystore.jks
listeners.https.ssl.keystore.password=test1234
listeners.https.ssl.key.password=test1234
```

Confluent Replicator

Confluent Replicator is a type of Kafka source connector that replicates data from a source to destination Kafka cluster. An embedded consumer inside Replicator consumes data from the source cluster, and an embedded producer inside the Kafka Connect worker produces data to the destination cluster.

Replicator version 4.0 and earlier requires a connection to ZooKeeper in the origin and destination Kafka clusters. If ZooKeeper is configured for authentication, the client configures the ZooKeeper security credentials via the global JAAS configuration setting `-Djava.security.auth.login.config` on the Connect workers, and the ZooKeeper security credentials in the origin and destination clusters must be the same.

To configure Confluent Replicator security, you must configure the Replicator connector as shown below and additionally you must configure:

- Kafka Connect

To add SSL to the Confluent Replicator embedded consumer, modify the Replicator JSON properties file.

Here is an example subset of configuration properties to add for SSL encryption:

```
{
  "name":"replicator",
    "config":{
      ....
      "src.kafka.bootstrap.servers" : "kafka1:9093",
      "src.kafka.security.protocol" : "SSL",
      "src.kafka.ssl.truststore.location" : "var/private/ssl/kafka.server.truststore.jks",
      "src.kafka.ssl.truststore.password" : "test1234",
      ....
    }
  }
}
```

Confluent Control Center

Confluent Control Center uses Kafka Streams as a state store, so if all the Kafka brokers in the cluster backing Control Center are secured, then the Control Center application also needs to be secured.

Enable security for the Control Center application as described in the section below. Additionally, configure security for the following components:

- Confluent Metrics Reporter: required on the production cluster being monitored
- Confluent Monitoring Interceptors: optional if you are using Control Center streams monitoring

Enable SSL for Control Center in the `etc/confluent-control-center/control-center.properties` file.

```
confluent.controlcenter.streams.security.protocol=SSL
confluent.controlcenter.streams.ssl.truststore.location=/var/private/ssl/kafka.client.truststore.jks
confluent.controlcenter.streams.ssl.truststore.password=test1234
```

Confluent Metrics Reporter

This section describes how to enable SSL encryption for Confluent Metrics Reporter, which is used for Confluent Control Center and Auto Data Balancer.

To add SSL for the Confluent Metrics Reporter, add the following to the `server.properties` file on the brokers in the Kafka cluster being monitored.

```
confluent.metrics.reporter.bootstrap.servers=kafka1:9093
confluent.metrics.reporter.security.protocol=SSL
confluent.metrics.reporter.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
confluent.metrics.reporter.ssl.truststore.password=test1234
```

Confluent Monitoring Interceptors

Confluent Monitoring Interceptors are used for Confluent Control Center streams monitoring. This section describes how to enable security for Confluent Monitoring Interceptors in three places:

1. General clients
2. Kafka Connect
3. Confluent Replicator

Interceptors for General Clients

For Confluent Control Center stream monitoring to work with Kafka clients, you must configure SSL encryption for the Confluent Monitoring Interceptors in each client.

1. Verify that the client has configured interceptors.

- Producer:

```
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor
```

- Consumer:

```
interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
```

2. Configure SSL encryption for the interceptor.

```
confluent.monitoring.interceptor.bootstrap.servers=kafka1:9093
confluent.monitoring.interceptor.security.protocol=SSL
confluent.monitoring.interceptor.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
confluent.monitoring.interceptor.ssl.truststore.password=test1234
```

Interceptors for Kafka Connect

For Confluent Control Center stream monitoring to work with Kafka Connect, you must configure SSL for the Confluent Monitoring Interceptors in Kafka Connect. Configure the Connect workers by adding these properties in `connect-distributed.properties`, depending on whether the connectors are sources or sinks.

- Source connector: configure the Confluent Monitoring Interceptors for SSL encryption with the `producer` prefix.

```
producer.interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringProducerInterceptor
producer.confluent.monitoring.interceptor.bootstrap.servers=kafka1:9093
producer.confluent.monitoring.interceptor.security.protocol=SSL
producer.confluent.monitoring.interceptor.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
producer.confluent.monitoring.interceptor.ssl.truststore.password=test1234
```

- Sink connector: configure the Confluent Monitoring Interceptors for SSL encryption with the `consumer` prefix.

```
consumer.interceptor.classes=io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor
consumer.confluent.monitoring.interceptor.bootstrap.servers=kafka1:9093
consumer.confluent.monitoring.interceptor.security.protocol=SSL
consumer.confluent.monitoring.interceptor.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
consumer.confluent.monitoring.interceptor.ssl.truststore.password=test1234
```

Interceptors for Replicator

For Confluent Control Center stream monitoring to work with Replicator, you must configure SSL for the Confluent Monitoring Interceptors in the Replicator JSON configuration file. Here is an example subset of configuration properties to add for SSL encryption.

```
{
  "name":"replicator",
    "config":{
      ....
      "src.consumer.group.id": "replicator",
      "src.consumer.interceptor.classes": "io.confluent.monitoring.clients.interceptor.MonitoringConsumerInterceptor",
      "src.consumer.confluent.monitoring.interceptor.bootstrap.servers": "kafka1:9093",
      "src.consumer.confluent.monitoring.interceptor.security.protocol": "SSL",
      "src.consumer.confluent.monitoring.interceptor.ssl.truststore.location": "/var/private/ssl/kafka.client.truststore.jks",
      "src.consumer.confluent.monitoring.interceptor.ssl.truststore.password": "test1234",
      ....
    }
  }
}
```

Schema Registry

Schema Registry uses Kafka to persist schemas, and so it acts as a client to write data to the Kafka cluster. Therefore, if the Kafka brokers are configured for security, you should also configure Schema Registry to use security. You may also refer to the complete list of Schema Registry configuration options (https://docs.confluent.io/current/schema-registry/docs/config.html).

Here is an example subset of `schema-registry.properties` configuration parameters to add for SSL encryption:

```
kafkastore.bootstrap.servers=SSL://kafka1:9093
kafkastore.security.protocol=SSL
kafkastore.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
kafkastore.ssl.truststore.password=test1234
```

REST Proxy

Securing Confluent REST Proxy with SSL encryption requires that you configure security between:

1. REST clients and the REST Proxy (HTTPS)
2. REST proxy and the Kafka cluster

You may also refer to the complete list of REST Proxy configuration options (https://docs.confluent.io/current/kafka-rest/docs/config.html#security-configuration-options).

First configure HTTPS between REST clients and the REST Proxy. Here is an example subset of `kafka-rest.properties` configuration parameters to configure HTTPS:

```
ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
ssl.truststore.password=test1234
```

Then, configure SSL encryption between REST proxy and the Kafka cluster. Here is an example subset of `kafka-rest.properties` configuration parameters to add for SSL encryption:

```
client.bootstrap.servers=kafka1:9093
client.security.protocol=SSL
client.ssl.truststore.location=/var/private/ssl/kafka.server.truststore.jks
client.ssl.truststore.password=test1234
```

SSL Logging

Enable SSL debug logging at the JVM level by starting the Kafka broker and/or clients with the `javax.net.debug` system property. For example:

```
export KAFKA_OPTS=-Djavax.net.debug=all
bin/kafka-server-start etc/kafka/server.properties
```

Tip

These instructions assume you are installing Confluent Platform by using ZIP or TAR archives. For more information, see On-Premises Deployments (../installation/installing_cp/index.html#installation).

Once you start the broker you should be able to see in the `server.log`:

```
with addresses: PLAINTEXT -> EndPoint(192.168.64.1,9092,PLAINTEXT),SSL -> EndPoint(192.168.64.1,9093,SSL)
```

To verify if the server's keystore and truststore are setup correctly you can run the following command:

```
openssl s_client -debug -connect localhost:9093 -tls1
```

Note: TLSv1 should be listed under `ssl.enabled.protocols`.

In the output of this command you should see the server's certificate:

```
-----BEGIN CERTIFICATE-----
{variable sized random bytes}
-----END CERTIFICATE-----
subject=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=Joe Smith
issuer=/C=US/ST=CA/L=Santa Clara/O=org/OU=org/CN=kafka/emailAddress=test@test.com
```

You can find more details on this in the Oracle documentation (http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/ReadDebug.html) on debugging SSL/TLS connections (http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/ReadDebug.html).

If the certificate does not show up with the `openssl` command, or if there are any other error messages, then your keys or certificates are not setup correctly. Review your configurations.
