**KAFKA Security – Moving away from PLAINTEXT to SASL\_SCRAM|SSL**

Moving from **SASL/PLAIN** to **SASL/SCRAM** is a strong step toward production-grade Kafka security. SCRAM hashes credentials with PBKDF2, stores them securely in ZooKeeper (or KRaft metadata), and eliminates the risk of transmitting raw passwords.

**Step 1: Configure Broker for SASL/SSL**

[**server.properties**](https://server.properties)

|  |
| --- |
| *listeners=SASL\_SSL://:9093*  *advertised.listeners=SASL\_SSL://your.host.name:9093*  *listener.security.protocol.map=SASL\_SSL:SASL\_SSL*  *inter.broker.listener.name=SASL\_SSL*  *sasl.mechanism.inter.broker.protocol=SCRAM-SHA-512*  *security.inter.broker.protocol=SASL\_SSL*  *ssl.keystore.location=/path/to/keystore.jks*  *ssl.keystore.password=keystore-password*  *ssl.key.password=key-password*  *ssl.truststore.location=/path/to/truststore.jks*  *ssl.truststore.password=truststore-password* |

**JAAS File (e.g. kafka\_server\_jaas.conf)**

|  |
| --- |
| *KafkaServer {*  *org.apache.kafka.common.security.scram.ScramLoginModule required*  *username="admin"*  *password="admin-secret";*  *};* |

This username/password is only for inter-broker auth. Client users will be stored in ZooKeeper or KRaft.

**Step 2: Add SCRAM Credentials to ZooKeeper**

Use the built-in [kafka-configs.sh](https://kafka-configs.sh) to provision users:

|  |
| --- |
| *# Create user 'alice' with password 'alice-password'*  *bin/kafka-configs.sh --zookeeper localhost:2181*  *--alter --add-config 'SCRAM-SHA-512=[password=alice-password]'*  *--entity-type users --entity-name alice*  *# (Optional) Add inter-broker user*  *bin/kafka-configs.sh --zookeeper localhost:2181*  *--alter --add-config 'SCRAM-SHA-512=[password=admin-secret]'*  *--entity-type users --entity-name admin* |

**If using KRaft mode, swap --zookeeper with --bootstrap-server and use --admin.**

**Step 3: Client-Side JAAS + Config**

**client\_jaas.conf**

|  |
| --- |
| *KafkaClient {*  *org.apache.kafka.common.security.scram.ScramLoginModule required*  *username="alice"*  *password="alice-password";*  *};* |

[**client.properties**](https://client.properties)

|  |
| --- |
| *security.protocol=SASL\_SSL*  *sasl.mechanism=SCRAM-SHA-512*  *ssl.truststore.location=/path/to/truststore.jks*  *ssl.truststore.password=truststore-password* |

**✅ Final Checklist**

|  |  |
| --- | --- |
| **Component** | **Status** |
| Broker JAAS | ✅ SCRAM module |
| ZooKeeper/KRaft | ✅ User credentials stored securely |
| Client JAAS | ✅ Password passed to broker |
| Wire Protocol | ✅ Encrypted with SSL |

**Enhancements Overview**

All of these steps can be automated. This is what we will be doing and why.

**1. Trigger** [**kafka-configs.sh**](https://kafka-configs.sh)

* Dynamically provision SCRAM credentials for inter-broker and client users.
* Supports both ZooKeeper and KRaft modes (auto-detectable via CLI flag or env).

**2. Cert Rotation Module**

* Generates new certs with a timestamped alias or alternate CN.
* Validates keystore/truststore integrity.
* Swaps certs atomically.
* Restarts brokers sequentially (safe rolling update).

**Single Executable Script**

Here’s a consolidated script that combines

* SCRAM provisioning
* cert rotation
* and mode detection

into a single executable

|  |
| --- |
| #!/bin/bash  # Configurable Variables  source .env  CLIENT\_USERS=(alice bob carol)  KAFKA\_MODE=zookeeper # or kraft  DOMAIN=example.com  KEYSTORE\_PASS=keystore-password  TRUSTSTORE\_PASS=truststore-password  CERT\_DAYS=365  # Function to provision SCRAM credentials  provision\_scram() {  local mode="$1" # zookeeper or kraft  local broker="$2" # e.g., localhost:9093  local zk="$3" # e.g., localhost:2181  for user in "${CLIENT\_USERS[@]}"; do  local cmd="bin/kafka-configs.sh"  if [[ "$mode" == "kraft" ]]; then  $cmd --bootstrap-server "$broker" \\  --alter --add-config "SCRAM-SHA-512=[password=${user}-pass]" \\  --entity-type users --entity-name "$user"  else  $cmd --zookeeper "$zk" \\  --alter --add-config "SCRAM-SHA-512=[password=${user}-pass]" \\  --entity-type users --entity-name "$user"  fi  echo "🔑 Provisioned SCRAM user: $user"  done  }  # Function to rotate certs  rotate\_certs() {  local id="$1"  local cn="kafka${id}.${DOMAIN}"  local old\_dir="secrets/kafka${id}"  local new\_dir="secrets/kafka${id}\_rotated\_$(date +%Y%m%d\_%H%M)"  mkdir -p "$new\_dir"  echo "🔁 Rotating certs for broker ${id}"  keytool -genkey -alias kafka -keyalg RSA -keystore "$new\_dir/kafka.keystore.jks" \\  -storepass "$KEYSTORE\_PASS" -keypass "$KEYSTORE\_PASS" -validity "$CERT\_DAYS" \\  -dname "CN=${cn}, OU=Kafka, O=Company, L=City, S=State, C=IN"  keytool -export -alias kafka -keystore "$new\_dir/kafka.keystore.jks" \\  -file "$new\_dir/kafka.cert" -storepass "$KEYSTORE\_PASS"  keytool -import -alias kafka -file "$new\_dir/kafka.cert" \\  -keystore "$new\_dir/kafka.truststore.jks" -storepass "$TRUSTSTORE\_PASS" -noprompt  echo "✔️ New certs generated in $new\_dir"  echo "🧪 Validating keystore..."  keytool -list -v -keystore "$new\_dir/kafka.keystore.jks" -storepass "$KEYSTORE\_PASS" | grep Owner  echo "📦 Backing up old certs..."  mv "$old\_dir" "${old\_dir}\_backup\_$(date +%s)"  echo "🚚 Deploying new certs to $old\_dir"  cp "$new\_dir"/\* "$old\_dir"/  echo "🔄 Restarting kafka${id} container..."  docker restart "kafka${id}"  }  # Function to template client-side JAAS  template\_client\_jaas() {  local user="$1"  local pass="$2"  cat <<EOF > "client\_jaas\_${user}.conf"  KafkaClient {  org.apache.kafka.common.security.scram.ScramLoginModule required  username="$user"  password="$pass";  };  EOF  echo "📜 Client JAAS template created for user: $user"  }  # Main Script  case "$1" in  --provision)  provision\_scram "$KAFKA\_MODE" "localhost:9093" "localhost:2181"  ;;  --rotate)  for id in {1..3}; do  rotate\_certs "$id"  done  ;;  --template)  for user in "${CLIENT\_USERS[@]}"; do  template\_client\_jaas "$user" "${user}-pass"  done  ;;  \*)  echo "Usage: $0 --provision|--rotate|--template"  ;;  esac |

This script provides:

* **--provision**: Provisions SCRAM credentials for all users in CLIENT\_USERS.
* **--rotate**: Rotates certs for brokers kafka1, kafka2, kafka3.
* **--template**: Generates client-side JAAS templates for all users in CLIENT\_USERS.

Feel free to extend it with additional flags or integrate it into your CI/CD pipeline for seamless Kafka security management.