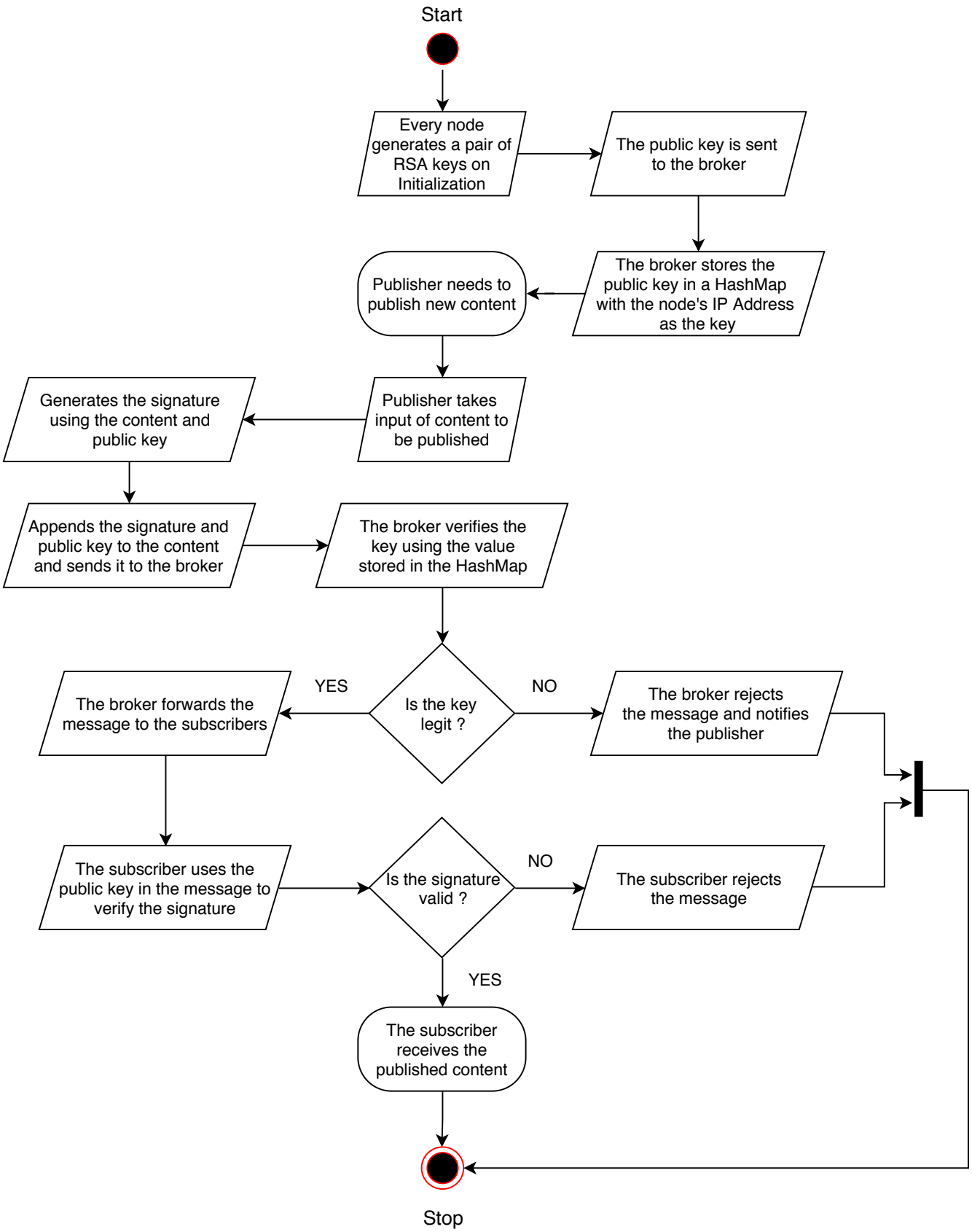
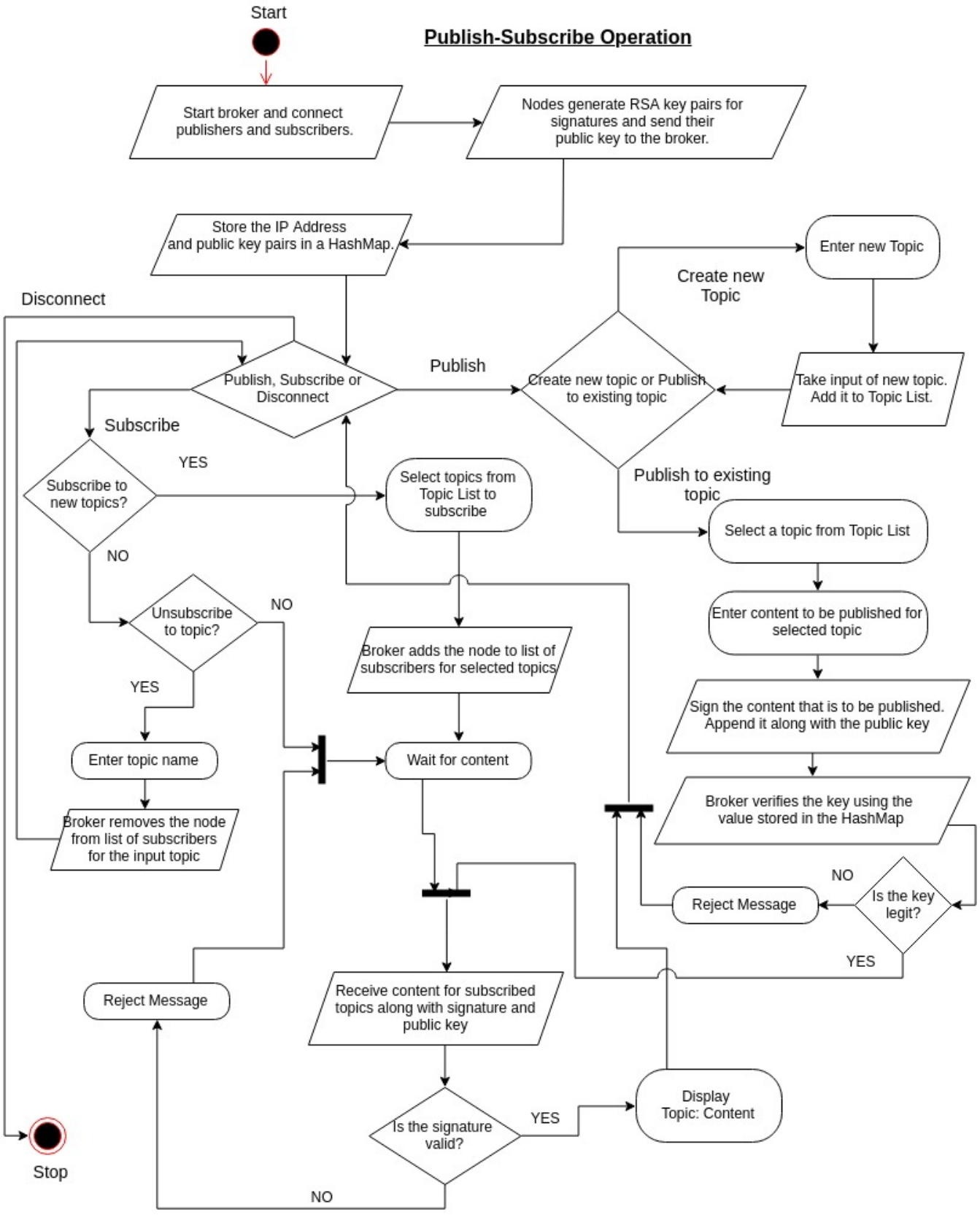
Flowchart for Handling Security




Start
Publish-Subscribe Operation
Start broker and connect publishers and subscribers.
Nodes generate RSA key pairs for signatures and send their public key to the broker.
Store the IP Address and public key pairs in a HashMap.
Enter new Topic
Create new Topic
Disconnect
Publish, Subscribe or Disconnect
Publish
Create new topic or Publish to existing topic
Take input of new topic. Add it to Topic List.
Subscribe
Subscribe to new topics?
YES
Select topics from Topic List to subscribe
Publish to existing topic
Select a topic from Topic List
NO
Unsubscribe to topic?
NO
Broker adds the node to list of subscribers for selected topics
Enter content to be published for selected topic
YES
Sign the content that is to be published. Append it along with the public key
Enter topic name
Wait for content
Broker verifies the key using the value stored in the HashMap
Broker removes the node from list of subscribers for the input topic
NO
Reject Message
Is the key legit?
YES
Reject Message
Receive content for subscribed topics along with signature and public key
Display Topic: Content
Is the signature valid?
YES
Stop
NO

#NODE CRASH:

SCENARIO:

- Accidently if any of the Subscriber crashes or goes down

IMPLEMENTED SOLUTION:

- Broker detects the failure and stores the data Published by Publishers against the failed Client/Node IP address
- Broker also keeps track of the Clients and the Topics they have subscribed till the date
- When the crashed Client/ Subscriber/ Node comes back in operation, it first retrieves it`s Missed data and the Snapshot of its subscription from Broker
- After this, it continuous to receive the Data for the Topics it has subscribed to and functions normally

#SECURITY:

SCENARIO:

- Hacking data by using invalid key or signature

IMPLEMENTED SOLUTION:

- When a node is connected, a new RSA-2048 key pair is generated. The public key is then shared with the broker.
- The broker stores this public key in a HashMap with the IP Address of the node as the key.
- When a publisher publishes new content, the signature is calculated using the RSA-2048 private key and SHA-256 and appended to the content along with the public key.
- The broker verifies the public key. **If invalid, rejects the message** and notifies the publisher. If valid, forwards it to the subscriber.
- The subscriber verifies the signature using the public key and if the signature is valid, the content is accepted. Else, the message is rejected.

#Workflow:

- Nodes connect to broker. Generate RSA key pairs. Send the public key to broker.

- **PUBLISH:**
  - Nodes choose to either publish or subscribe. If they choose to publish, enters data for topic to publish or create a new topic.
  - Publisher signs the content before publishing it and sends it along with the message and public key. The broker verifies the public key. If valid, forwards it to the subscriber. The subscriber verifies the signature.

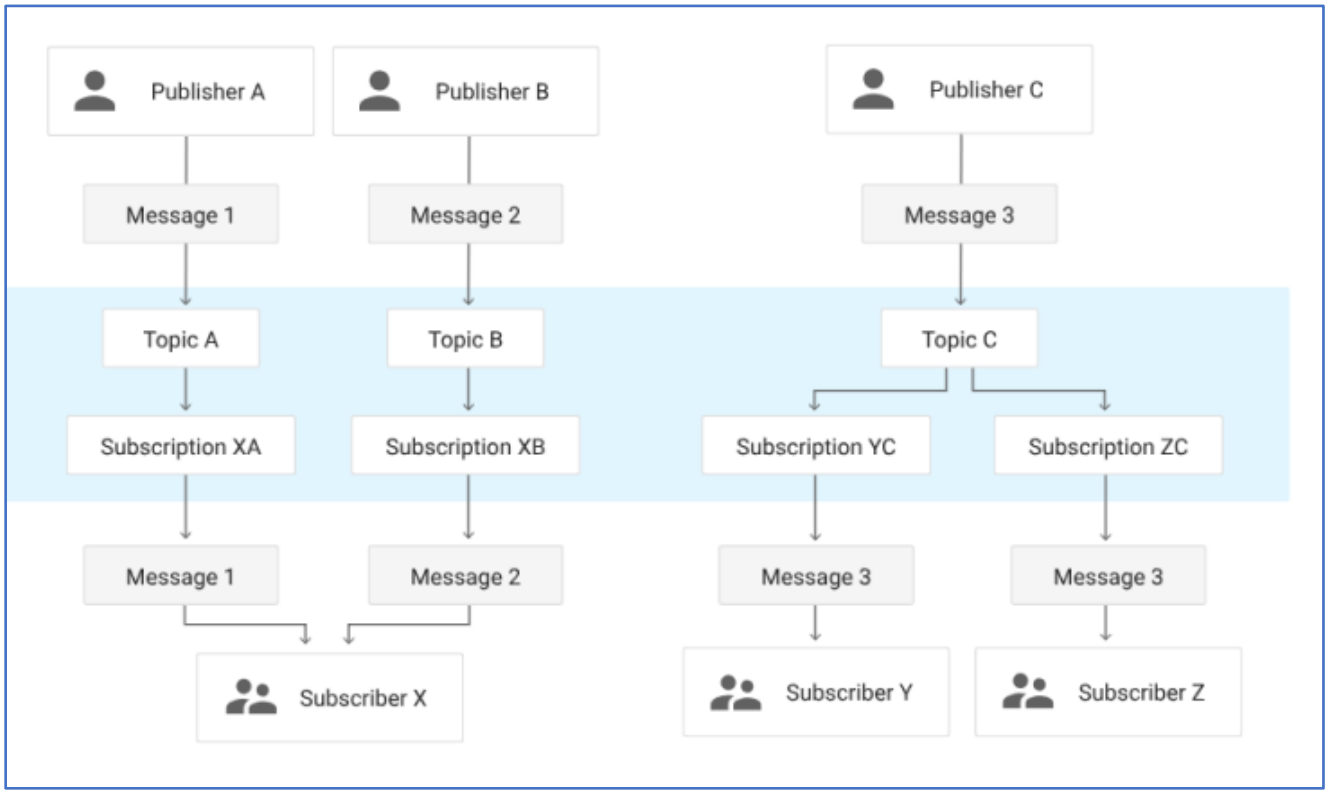
- **SUBSCRIBE:**
  - If the node chooses to subscribe, they can choose topics to subscribe to, from the list of topics provided by the broker. When new content is published under the subscribed topics, after verifying the signature, the subscriber displays the content.

- **UN-SUBSCRIBE:**
  - Client choses to un-subscribe from one of the Topic and stops receiving content of that particular topic in future.

- **RESET:**
  - Client totally disconnects from the network and loses all its subscriptions. Receives no more data.

Secured Publisher-Subscriber System

- It is an **asynchronous messaging model** where Subscribers can subscribe to their Topics of interest. Publishers can publish data for multiple topics; and, subscribers will receive the data for Topics they have subscribed to.
- Broker is like a medium that facilitates the flow of the data with efficient speed and takes care of heavy operations like **handling multiple transactions, connecting with database, handling node failure, persisting snapshot of subscribers** etc.
- Publishers and Subscribers have no knowledge of each other and are **loosely coupled.**
- Use case: Sensor network, Micro-services, Event notification



Our Implementation

- Implemented **Simplified MQTT** protocol for standardized communication between clients/ nodes and the broker.
- Multi-threading for handling each separate TCP communication channels between Broker and clients
- Using **RSA-2048 with SHA-256** signature algorithm to sign the content being published to preserve message integrity.
- Deployed on **100 nodes** on Google cloud to check the scalability performance testing
- Handled Node crash scenario by preserving missed data and subscribers snapshot

Outcome

- Tested the system for 100 nodes on Google cloud and the latency was on average **143.28ms** with **100%** message delivery.
- Successfully tested the node crash failure where Subscriber received all the missed messages.
- Successfully tested the secured data flow by modifying the keys and tampering with the signature.

Future Improvements

- Broker can be made distributed to manage the large number of requests efficiently
- The performance can further be improved with more optimized and careful multithreading design
- Can be made more secure by encrypting the entire content and then signing it with the overhead of the problem of key sharing.

Node or System Crash Scenario