The Anonymous Pub-Sub System:
A Documentation

Students:
Manasik Hassan
Nadia Charef
Nafisah Abdulkadir
Nikita Punnoose

1. Introduction:

Distributed systems and applications are preferred due to many advantages it brings. However, due to the unique approach of the system architecture & how information are maintained within it, security within distributed systems is of considerable importance.

Objectives:

- To implement a special kind of distributed publisher-subscriber system enabling two or more clients to communicate anonymously.
- Ensuring the anonymity of data communications by encrypting data with randomly generated ID before sending it & using that ID at the receiver side to retrieve it.

2. System Architecture:

In this section we will describe our technical approach to implement the anonymous pub-sub system.

2.1. Technologies we used:

Backend:

Python programming language was used to implement all the backend functionalities of the system because of its simplicity and its support for many standard libraries, Python was used to implement the following functionalities:

- To generate a random ID of 1000 bits long.
- To Encrypt and decrypt the data chunks with the generated ID.
- To perform the similarity check.
- To store data to the data bank.
- Flask library was used to build the web application.

Frontend (Nafisah):
Javascript (JQuery library), HTML and CSS was used to implement the frontend functionalities of the system.

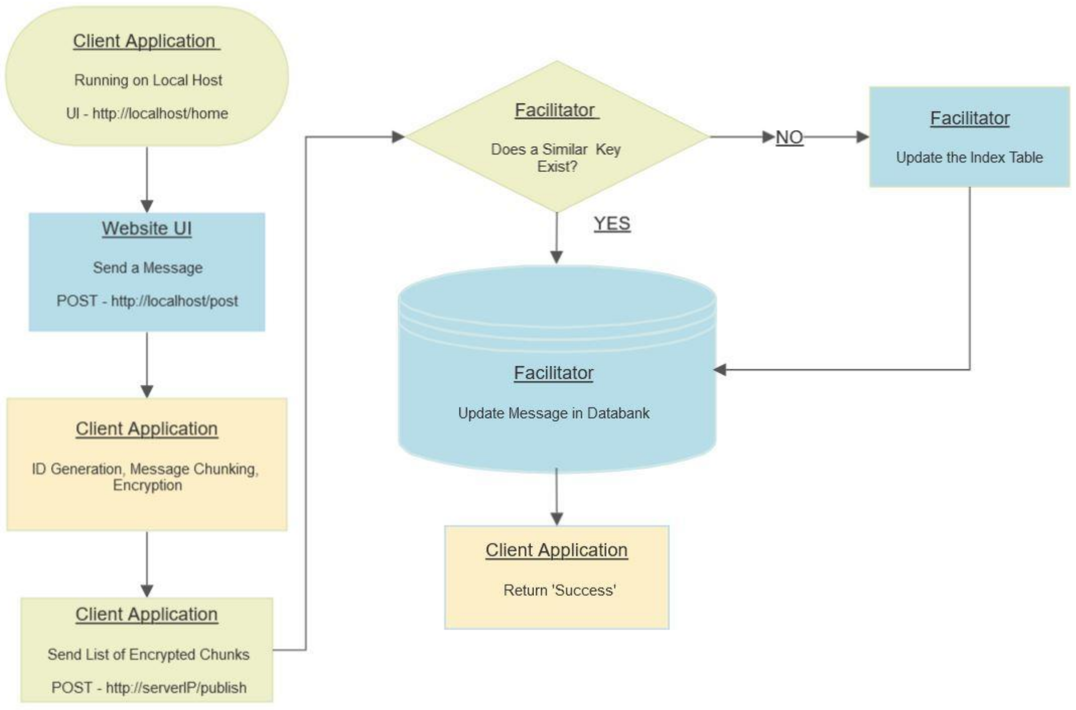
Data Storage:

Data storage is implemented by using a python dictionary as an autoassociative table which is linked to a csv file that acts as a data bank.

2.2. System Functionalities:

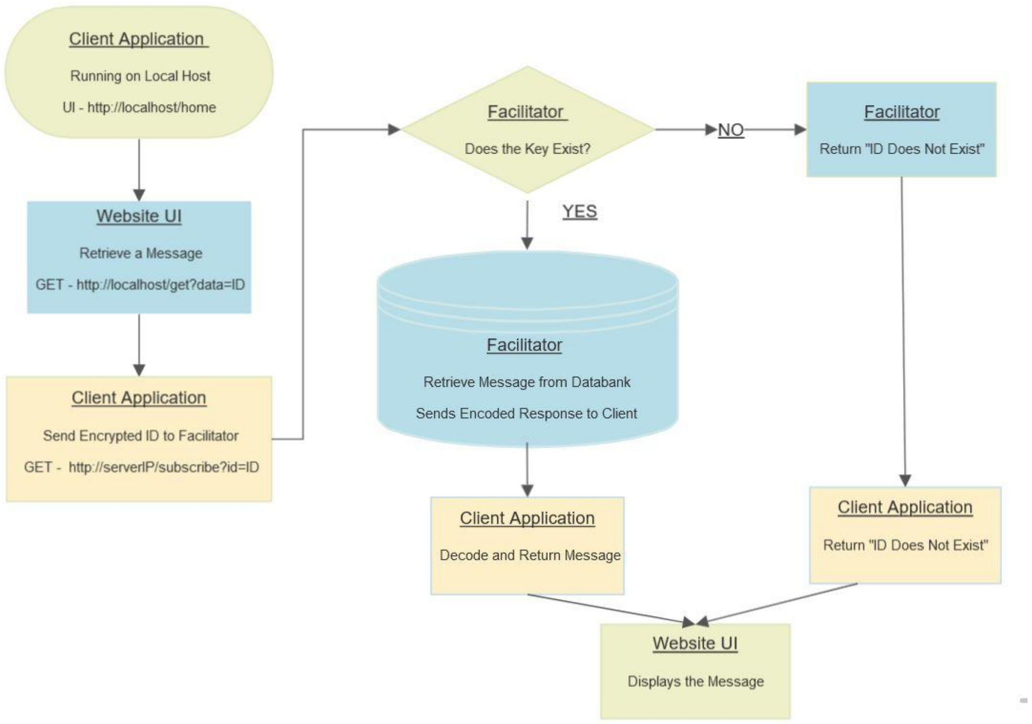
<u>Publish a message to the facilitator:</u>

- **Generate a random ID:** when the client connects to the server and sends its first message, a unique bipolar ID is generated to it.
- **Send a message to the network facilitator:** the message divided into chunks of 1000 characters. Each chunk can take a maximum of 142 characters (7 x 142 = 994 bits), the rest of the bits are padded with bits of ones.
- **Encryption of Data:** each chunk is encrypted with the user ID according to the following formula Data* = (Data x ID) + ID, where Data* is the encrypted data.
- **Similarity check at the facilitator:** the facilitator checks if the autoassociative table has entries. If the autoassociative table is empty, the first chunk is added to the autoassociative table and the data is populated to the data bank.
  Otherwise, it performs a similarity check between the first chunk of received data and the existing keys first chunks of other stored messages – using the dot product. We identified a threshold of 0.028. If the maximum of the obtained results from the dot product is greater than 0.028, the key exists and data corresponding to this key in the data bank (csv file) is updated.
  If the maximum of the obtained results from the dot product is less than 0.028, the key does not exist. The first chunk of the received message is appended to the autoassociative table and the data is populated to the data bank.
- The flowchart below depicts the steps involved to publish a message to the facilitator.


Client Application
Running on Local Host
UI - http://localhost/home
Website UI
Send a Message
POST - http://localhost/post
Client Application
ID Generation, Message Chunking, Encryption
Client Application
Send List of Encrypted Chunks
POST - http://serverIP/publish
Facilitator
Does a Similar Key Exist?
NO
Facilitator
Update the Index Table
YES
Facilitator
Update Message in Databank
Client Application
Return 'Success'

Retrieve a message from the facilitator:

- **Encryption of ID:** the ID of the client-sender is encrypted using a randomly generated sequence of 1 and -1.
- **The network facilitator:** the network facilitator performs a similarity check for the received ID to check if the ID is valid. If the ID is valid, we compute the dot product between the received ID and the keys in the autoassociative table. If the maximum of the obtained results from the dot product is greater than 0.028, the data corresponding to the maximum result of the dot product is retrieved and sent back to the client-recipient. Otherwise, an invalid key response is sent to the client-recipient.
- **Decode the received data:** the received data is decoded using the ID of the client-sender.
- The flowchart below depicts the steps involved to retrieve a message from the facilitator.


Client Application
Running on Local Host
UI - http://localhost/home
Website UI
Retrieve a Message
GET - http://localhost/get?data=ID
Client Application
Send Encrypted ID to Facilitator
GET - http://serverIP/subscribe?id=ID
Facilitator
Does the Key Exist?
NO
Facilitator
Return "ID Does Not Exist"
YES
Facilitator
Retrieve Message from Databank
Sends Encoded Response to Client
Client Application
Decode and Return Message
Client Application
Return "ID Does Not Exist"
Website UI
Displays the Message