



# Information Security

## Lab Assignment I

Aarón Cela Riveiro

In this lab assignment, I developed an application that securely encrypts, sends and receives messages using nested hybrid encryption and onion routing over an MQTT broker. Messages are routed through multiple hops, each decrypting only one layer before forwarding it, preserving privacy and sender anonymity. The project is divided into three modules: `tor.py` for encryption logic, `mqtt-listener.py` for message relaying, and `mqtt-sender.py` for sending routed messages. An additional file, `pubkeys.py`, stores public keys of classmates for testing multi-hop communication.

Full implementation available at <https://github.com/ronezz/SI/tree/main/LAB1>.

### File: `tor.py`

This file contains the core cryptographic functions of the project. Below, each group of functions is explained and shown in code blocks.

#### 1. Key Management

Open my private and public keys (read from `private_key.pem` and `public_key.pub`):

```
1 from cryptography.hazmat.backends import default_backend
2 from cryptography.hazmat.primitives.asymmetric import rsa
3 from cryptography.hazmat.primitives import serialization
4 from cryptography.hazmat.primitives import hashes
5 from cryptography.hazmat.primitives.asymmetric import padding
6 import os
7 from cryptography.hazmat.primitives.ciphers.aead import AESGCM
8 import pubkeys
9
10
11 # Open private key
12 with open("private_key.pem", "rb") as key_file:
13     private_key = serialization.load_ssh_private_key(
14         key_file.read(),
15         password=None,
16         backend=default_backend()
17     )
18
19 # Open public key
20 with open("public_key.pub", "rb") as key_file:
21     public_key = serialization.load_ssh_public_key(
22         key_file.read(),
23         backend=default_backend()
24     )
```

Listing 1: Read RSA private and public keys

Normalize user identifiers and convert back from bytes to string:

```
1 # Format userId to 5b
2 def adjust_userId(user_id):
3     b = user_id.encode('utf-8')
4     b = b[:5]
5     b = b.ljust(5, b'\x00')
6     return b
7
8 def _bytes5_to_str(b5):
9     return b5.rstrip(b'\x00').decode('utf-8', errors='ignore')
```

Listing 2: User-id helpers

Look up a recipient's public key from the shared dictionary:

```
1 # Find user public key
2 def find_pubKey(user_id):
3     public_key= pubkeys.pubkey_dict.get(user_id)
4
5     if public_key is not None:
6         return serialization.load_ssh_public_key(
7             ('ssh-rsa ' + public_key).encode('ascii'),
8             backend=default_backend())
9     else:
10        print("Public key not found")
11
12    return public_key
```

Listing 3: Find recipient public key

## 2. Hybrid Encryption

AES-GCM functions for Encrypt/Decrypt

```
1 # AESGCM Functions
2 def aesgcm_encryption(key, data):
3     aesgcm = AESGCM(key)
4     nonce = key
5     ciphertext = aesgcm.encrypt(nonce, data, None)
6     return ciphertext
7
8
9 def aesgcm_decryption(key, ciphertext):
10    aesgcm = AESGCM(key)
11    nonce = key
12    return aesgcm.decrypt(nonce, ciphertext, None)
```

Listing 4: AES-GCM helpers

RSA-OAEP functions for encrypt/decrypt

```
1 # RSA Functions
2 def rsa_encryption(public_key, message):
3     encrypted = public_key.encrypt(
4         message,
5         padding.OAEP(
6             mgf=padding.MGF1(algorithm=hashes.SHA256()),
7             algorithm=hashes.SHA256(),
8             label=None)
9     )
10    return encrypted
11
12
```

```

13 def rsa_decryption(encrypted):
14     original_message = private_key.decrypt(
15         encrypted,
16         padding.OAEP(
17             mgf=padding.MGF1(algorithm=hashes.SHA256()),
18             algorithm=hashes.SHA256(),
19             label=None)
20     )
21     return original_message

```

Listing 5: RSA-OAEP helpers

#### Hybrid wrappers

```

1  # Hybrid Encryption/Decryption
2  def hybrid_encryption(public_key, data):
3      key = AESGCM.generate_key(bit_length=128)
4      aescgm_cipher = aescgm_encryption(key, data)
5      rsa_cipher = rsa_encryption(public_key, key)
6      return rsa_cipher + aescgm_cipher
7
8  def hybrid_decryption(ciphertext):
9      key_length = private_key.key_size // 8
10
11     cipher_key = ciphertext[:key_length]
12     cipher_data = ciphertext[key_length:]
13
14     key = rsa_decryption(cipher_key)
15     data = aescgm_decryption(key, cipher_data)
16
17     return data

```

Listing 6: Hybrid encryption/decryption

### 3. Onion Layer Construction

Build the onion

```
1  # Build
2  def nest_hybrid_encryption(path, data, anonymous):
3      sender = path[0]
4      recipient = path[-1]
5
6      # Check if we want to send the message anonymously
7      sender_field = "none" if anonymous else sender
8
9      final_content = adjust_userId("end") + adjust_userId(sender_field) + data
10     ciphertext = hybrid_encryption(find_pubKey(recipient), final_content)
11     for index in range(len(path) - 2, -1, -1):
12         current_hop = path[index]
13         next_hop = path[index + 1]
14         payload_for_hop = adjust_userId(next_hop) + ciphertext
15         ciphertext = hybrid_encryption(find_pubKey(current_hop),
16                                         payload_for_hop)
17
18     return ciphertext
```

Listing 7: Nested onion construction

### Decryption and Relay

8- Peel the onion

```
1  # Peel
2  def decode_and_relay(ciphertext):
3
4      payload = hybrid_decryption(ciphertext)
5      next_hop = payload[:5]
6      rest = payload[5:]
7
8      nh = _bytes5_to_str(next_hop).lower()
9      if nh != "end":
10         return ("forward", _bytes5_to_str(next_hop), rest)
11     else:
12         sender = _bytes5_to_str(rest[:5])
13         message = rest[5:]
14         return ("deliver", sender, message)
```

Listing 8: Decode one layer and relay/deliver

## File: mqtt\_listener.py

This file is responsible for receiving and forwarding encrypted messages through the MQTT broker.

### 1. MQTT Configuration

The MQTT server parameters are defined here, as well as the identity of the current node, which is used to subscribe to inbound messages in its topic.

```
1 from paho.mqtt import client as mqtt_client
2 import tor
3
4 MQTT_SERVER = "18.101.140.151"
5 MQTT_USER = "sinf"
6 MQTT_PASSWORD = "sinf2025"
7 MQTT_PORT = 1883
8 MQTT_KEEPALIVE = 60
9
10 MY_ID = "ancr"
```

Listing 9: MQTT connection parameters

### 2. Message Callback Handler

This function is triggered automatically when a message is received on the MQTT topic associated with this node. It decodes one onion layer using `decode_and_relay()` from `tor.py`.

```
1 # While Listening...
2 def on_message(client, userdata, msg):
3     try:
4         action, who, payload = tor.decode_and_relay(msg.payload)
5         if action == "forward":
6             print(f"[{MY_ID}] Forwarding next hop '{who}'...")
7             client.publish(who, payload)
8         else:
9             try:
10                texto = payload.decode("utf-8")
11            except:
12                texto = repr(payload)
13            print(f"[{MY_ID}] Message from '{who}': {texto}")
14    except Exception as e:
15        print(f"[{MY_ID}] Error processing message: {e}")
```

Listing 10: Handle incoming message

### 3. MQTT Loop Initialization

This function connects to the MQTT broker, subscribes to the node's topic (its user-id), and waits indefinitely for incoming messages.

```
1 def main():
2     c = mqtt_client.Client()
3     c.username_pw_set(MQTT_USER, MQTT_PASSWORD)
4     c.on_message = on_message
5     c.connect(MQTT_SERVER, MQTT_PORT, MQTT_KEEPALIVE)
6     c.subscribe(MY_ID)
7     print(f"[{MY_ID}] Listening on topic '{MY_ID}'...")
8     c.loop_forever()
9
10 if __name__ == "__main__":
11     main()
```

## File: mqtt\_sender.py

This file is responsible for sending encrypted messages using nested hybrid encryption over MQTT. It uses the function `nest_hybrid_encryption()` from `tor.py` to wrap the message with multiple encryption layers before publishing it to the MQTT broker.

### 1. MQTT Configuration and Client Creation

The MQTT server credentials are defined here. A helper function is used to initialize and connect the MQTT client.

```

1 from paho.mqtt import client as paho_mqtt
2 import tor
3
4 MQTT_SERVER = "18.101.140.151"
5 MQTT_USER = "sinf"
6 MQTT_PASSWORD = "sinf2025"
7 MQTT_PORT = 1883
8 MQTT_TOPIC = "ancr"
9 MQTT_KEEPAIVE = 60
10
11 def mqtt_client(server, port, topic, user, password, keepalive):
12     client = paho_mqtt.Client()
13     client.username_pw_set(user, password)
14     client.connect(server, port, keepalive)
15     client.subscribe(topic)
16     return client
17
18
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER,
    MQTT_PASSWORD, MQTT_KEEPAIVE)

```

Listing 12: MQTT setup and client creation

### 2. Encrypt and Send a Message

A simple test message is encrypted using nested onion encryption. The ciphertext is then published to the first hop in the route.

```

1 m = b"Testing the network..."
2
3 path=["ancr","ancr","ancr","ancr","ancr"]
4
5 encrypted_to_send = tor.nest_hybrid_encryption(path, m, False)
6 result = client.publish("ancr", encrypted_to_send)

```

Listing 13: Encrypt and publish nested ciphertext

## Tests and Validation

This section presents a series of tests carried out to verify the correct operation of the implemented system, including nested hybrid encryption, onion-routing behaviour, and MQTT message delivery across multiple hops. The objective is to demonstrate that ciphertexts are successfully routed through one or more intermediate nodes and correctly decrypted only at the final destination.

### Test 1: Self-routing with multiple hops

In this first test, a message was sent from my user `ancr` to myself using a route with multiple self-intermediate hops.

```
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER, MQTT_PASSWORD, MQTT_KEEPALIVE)
21
22 m = b"Testing the network..."
23
24 path=["ancr","ancr","ancr","ancr","ancr"]
25
26 encrypted_to_send = tor.nest_hybrid_encryption(path, m, False)
27 result = client.publish("ancr", encrypted_to_send)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1> & C:\Users\aaaron\AppData\Local\Programs\Python\Python312\python.exe c:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1\mqtt-listener.py  
c:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1\mqtt-listener.py:29: DeprecationWarning: Call  
back API version 1 is deprecated, update to latest version  
c = mqtt\_client.Client()  
[ancr] Listening on topic 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Message from 'ancr': Testing the network...

PS C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1> python .\mqtt-sender.py  
C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1\mqtt-sender.py:12: DeprecationWarning: Callb  
ack API version 1 is deprecated, update to latest version  
client = paho\_mqtt.Client()  
PS C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1>

Figure 1: Self-message routing through multiple nested hops

```
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER, MQTT_PASSWORD, MQTT_KEEPALIVE)
21
22 m = b"Testing the network anonymously..."
23
24 path=["ancr","ancr","ancr","ancr","ancr"]
25
26 encrypted_to_send = tor.nest_hybrid_encryption(path, m, True)
27 result = client.publish("ancr", encrypted_to_send)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1> & C:\Users\aaaron\AppData\Local\Programs\Python\Python312\python.exe c:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1\mqtt-listener.py  
c:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1\mqtt-listener.py:29: DeprecationWarning: Call  
back API version 1 is deprecated, update to latest version  
c = mqtt\_client.Client()  
[ancr] Listening on topic 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Forwarding next hop 'ancr'...  
[ancr] Message from 'none': Testing the network anonymously...

PS C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1> python .\mqtt-sender.py  
C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1\mqtt-sender.py:12: DeprecationWarning: Callb  
ack API version 1 is deprecated, update to latest version  
client = paho\_mqtt.Client()  
PS C:\Users\aaaron\Documents\VMUNICS2\SI\Practicas\LAB-1>

Figure 2: Anonymous self-message routing through multiple nested hops

## Test 2: Long message delivery

This second test evaluates the system's ability to handle large messages exceeding a single RSA block. Thanks to the implemented hybrid encryption (RSA + AES-GCM), messages of arbitrary length can be securely transmitted.

```
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER, MQTT_PASSWORD, MQTT_KEEPALIVE)
21
22 m = b"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget augue sit amet dui blandit mattis quis quis tortor. Aliquam varius augue vitae neque dictum, dapibus
23
24 path=["ancr","ancr","ancr","ancr","ancr"]
25
26 encrypted_to_send = tor.nest_hybrid_encryption(path, m, False)
27 result = client.publish("ancr", encrypted_to_send)
```

```
c = mqtt_client.Client()
[ancr] Listening on topic 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Message from 'ancr': Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget augue sit amet dui blandit mattis quis quis tortor. Aliquam varius augue vitae neque dictum, dapibus imperdiet tellus vehicula. Aliquam quis lectus vitae dui consectetur sagittis in ut orci. Donec magna turpis, fringilla eget tellus sed, vestibulum finibus felis. Maecenas iaculis risus id nisi fringilla malesuada. Morbi vitae massa et orci volutpat facilisis et vel sapien. Sed finibus mollis arcu, non aliquam nunc blandit eu. Praesent luctus, dolor eu imperdiet faucibus, nibh justo sodales ipsum, ac efficitur arcu tellus a libero. Mauris in diam sit amet lacus egestas rhoncus eu sed nibh. Aliquam a laoreet neque. In hac habitasse platea dictumst. In hac habitasse platea dictumst. Duis sed risus nec augue lobortis luctus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultrices dui odio, eget fringilla nibh mollis id. Duis tincidunt lorem varius mollis hendrerit. Aenean quis neque pharetra elit bibendum luctus eleifend lacinia ex. Pellentesque urna lectus, blandit non ipsum quis, auctor lobortis est. Proin consectetur sapien in ligula fringilla aliquam. Donec in elit nunc. Aliquam quis finibus purus, sit amet bibendum urna. Nunc a orci a magna placerat vulputate a id velit. Phasellus pellentesque nisl eu luctus volutpat. Morbi porta congue metus, ut tempus justo porta quis. Vestibulum vitae sem velit. Proin eu tortor id odio consequat ultricies. Fusce quis sagittis nibh. Nullam ullamcorper ipsum a justo consectetur, interdum tristique arcu bibendum. Aliquam et odio sit amet nunc semper sagittis congue vel sapien. Integer sagittis eu massa id eleifend. Pellentesque feugiat dolor sed orci dignissim malesuada. Ut suscipit dapibus dapibus. Integer et dui id sem lobortis euismod. Integer interdum, purus in tincidunt pulvinar, risus velit rhoncus turpis, ut finibus tellus diam a libero. Maecenas porta pellentesque sapien vel vestibulum. Cras id dapibus augue. Suspendisse iaculis leo ut lorem gravida, vitae dictum nunc pellentesque. Vestibulum a risus ac nunc vestibulum convallis volutpat non arcu. Duis ut tincidunt velit. Aliquam massa est, sodales vel lectus mollis, aliquam mattis massa. Nullam
```

Figure 3: Successful routing of a large self-message through multiple nodes

```
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER, MQTT_PASSWORD, MQTT_KEEPALIVE)
21
22 m = b"Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget augue sit amet dui blandit mattis quis quis tortor. Aliquam varius augue vitae neque dictum, dapibus
23
24 path=["ancr","ancr","ancr","ancr","ancr"]
25
26 encrypted_to_send = tor.nest_hybrid_encryption(path, m, True)
27 result = client.publish("ancr", encrypted_to_send)
```

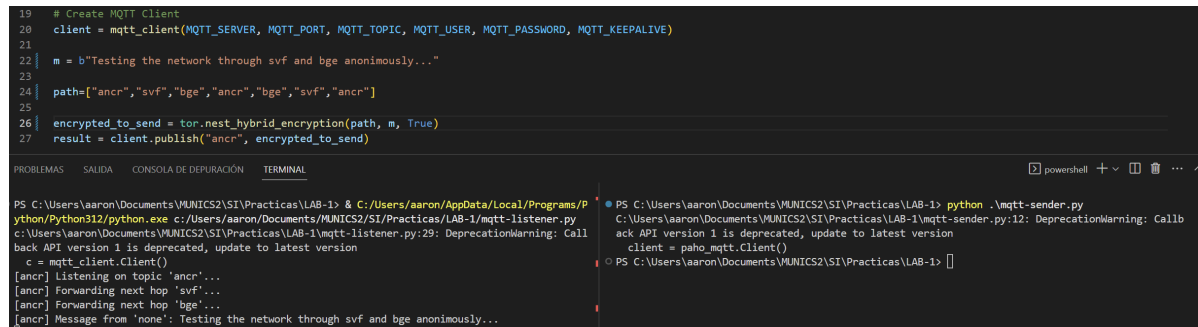
```
PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1> python .\mqtt-sender.py
C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1> python .\mqtt-listener.py
[ancr] Listening on topic 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Forwarding next hop 'ancr'...
[ancr] Message from 'none': Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam eget augue sit amet dui blandit mattis quis quis tortor. Aliquam varius augue vitae neque dictum, dapibus imperdiet tellus vehicula. Aliquam quis lectus vitae dui consectetur sagittis in ut orci. Donec magna turpis, fringilla eget tellus sed, vestibulum finibus felis. Maecenas iaculis risus id nisi fringilla malesuada. Morbi vitae massa et orci volutpat facilisis et vel sapien. Sed finibus mollis arcu, non aliquam nunc blandit eu. Praesent luctus, dolor eu imperdiet faucibus, nibh justo sodales ipsum, ac efficitur arcu tellus a libero. Mauris in diam sit amet lacus egestas rhoncus eu sed nibh. Aliquam a laoreet neque. In hac habitasse platea dictumst. In hac habitasse platea dictumst. Duis sed risus nec augue lobortis luctus. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nullam ultrices dui odio, eget fringilla nibh mollis id. Duis tincidunt lorem varius mollis hendrerit. Aenean quis neque pharetra elit bibendum luctus eleifend lacinia ex. Pellentesque urna lectus, blandit non ipsum quis, auctor lobortis est. Proin consectetur sapien in ligula fringilla aliquam. Donec in elit nunc. Aliquam quis finibus purus, sit amet bibendum urna. Nunc a orci a magna placerat vulputate a id velit. Phasellus pellentesque nisl eu luctus volutpat. Morbi porta congue metus, ut tempus justo porta quis. Vestibulum vitae sem velit. Proin eu tortor id odio consequat ultricies. Fusce quis sagittis nibh. Nullam ullamcorper ipsum a justo consectetur, interdum tristique arcu bibendum. Aliquam et odio sit amet nunc semper sagittis congue vel sapien. Integer sagittis eu massa id eleifend. Pellentesque feugiat dolor sed orci dignissim malesuada. Ut suscipit dapibus dapibus. Integer et dui id sem lobortis euismod. Integer interdum, purus in tincidunt pulvinar, risus velit rhoncus turpis, ut finibus tellus diam a libero. Maecenas porta pellentesque sapien vel vestibulum. Cras id dapibus augue. Suspendisse iaculis leo ut lorem gravida, vitae dictum nunc pellentesque. Vestibulum a risus ac nunc vestibulum convallis volutpat non arcu. Duis ut tincidunt velit. Aliquam massa est, sodales vel lectus mollis, aliquam mattis massa. Nullam leo dui, mattis quis suscipit in, consectetur ut ipsum. Suspendisse elementum nibh arcu, a euismod
```

Figure 4: Anonymous routing of a large self-message through multiple nodes



### Test 3: Routing through classmates

In the final test, a route was configured including two classmates, `svr` and `bge`, as intermediate relays.



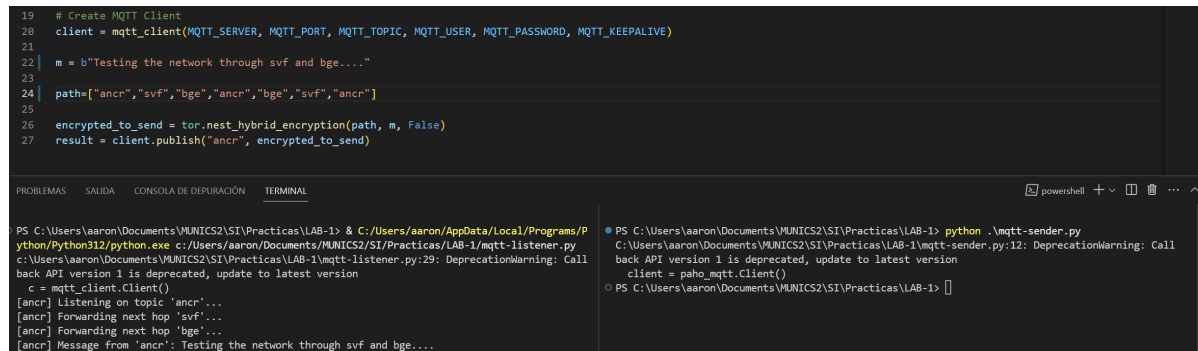
```
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER, MQTT_PASSWORD, MQTT_KEEPALIVE)
21
22 m = b"Testing the network through svf and bge anonymously..."
23
24 path=["ancr","svf","bge","ancr","bge","svf","ancr"]
25
26 encrypted_to_send = tor.nest_hybrid_encryption(path, m, True)
27 result = client.publish("ancr", encrypted_to_send)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1> & C:\Users\aaaron\AppData\Local\Programs\Python\Python312\python.exe c:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1\mqtt-listener.py  
c:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1\mqtt-listener.py:29: DeprecationWarning: Call back API version 1 is deprecated, update to latest version  
c = mqtt\_client.Client()  
[ancr] listening on topic 'ancr'...  
[ancr] Forwarding next hop 'svf'...  
[ancr] Forwarding next hop 'bge'...  
[ancr] Message from 'none': Testing the network through svf and bge anonymously...

PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1> python .\mqtt-sender.py  
C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1\mqtt-sender.py:12: DeprecationWarning: Call back API version 1 is deprecated, update to latest version  
client = paho\_mqtt.Client()  
PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1>

Figure 5: Anonymous message routed through classmates `svr` and `bge`



```
19 # Create MQTT Client
20 client = mqtt_client(MQTT_SERVER, MQTT_PORT, MQTT_TOPIC, MQTT_USER, MQTT_PASSWORD, MQTT_KEEPALIVE)
21
22 m = b"Testing the network through svf and bge...."
23
24 path=["ancr","svf","bge","ancr","bge","svf","ancr"]
25
26 encrypted_to_send = tor.nest_hybrid_encryption(path, m, False)
27 result = client.publish("ancr", encrypted_to_send)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1> & C:\Users\aaaron\AppData\Local\Programs\Python\Python312\python.exe c:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1\mqtt-listener.py  
c:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1\mqtt-listener.py:29: DeprecationWarning: Call back API version 1 is deprecated, update to latest version  
c = mqtt\_client.Client()  
[ancr] listening on topic 'ancr'...  
[ancr] Forwarding next hop 'svf'...  
[ancr] Forwarding next hop 'bge'...  
[ancr] Message from 'ancr': Testing the network through svf and bge....

PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1> python .\mqtt-sender.py  
C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1\mqtt-sender.py:12: DeprecationWarning: Call back API version 1 is deprecated, update to latest version  
client = paho\_mqtt.Client()  
PS C:\Users\aaaron\Documents\MUNICS2\SI\Practicas\LAB-1>

Figure 6: Standard message routed through classmates `svr` and `bge`