



SECURE MESSAGING

[Group Assignment -1]



CS6/745: Modern Cryptography

Submitted to: Dr. Yuliang Zheng

Submitted by:

1. Joyanta Mondal (jmondal)
2. Shouzab Khan (Skhan6)
3. Pooja Srinivas (psriniva)
4. Rajendra Mohan (rnavulur)

February 18, 2024

INTRODUCTION:

This report explains the implementation of a Python-based secure text exchange software package emphasizing socket communication by utilizing Ascon lightweight cryptographic algorithms for encryption and decryption. This program facilitates User A and User B a platform where they can safely communicate with one another across a network.

Your strategies for the design and implementation of the program:

Our strategy for design and implementation of this program was that first we have to design the communication between two computers in which one computer will run a server.py file and another will run a client.py file. For this purpose we need socket programming. For this python was used to establish a proper communication channel between server and client for sending encrypted messages.

Now when the channel is established, we have to send encrypted messages between server and client so for that purpose we need to secure our channel and for that we will use “ascon” lightweight cipher in python. “Pyascon” package is used in python for implementation of AEAD encryption and decryption of messages. The server and client will share a secret key, nonce, and associated data for secure communication.

Now, when both “Socket Programming” and “Encryption and Decryption” are done you have created a secure messaging platform. In which the sender will send an encrypted message to the receiver and receiver will decrypt it.

Part 1:

Here, we initially design the server.py and client.py in such a way that it can ensure sending messages to one and another.

Step 1: Open two terminals and go to the location where your client.py and server.py files are saved.

Step 2: Then, in one terminal, start server.py.

```
python -u "/Users/joyanta/UAB PhD/Spring 24/CS745/Assignment 1/Part 1 - JJ/server.py"
(py312) joyanta@Joyantas-MacBook-Pro ~ % python -u "/Users/joyanta/UAB PhD/Spring 24/CS745/Assignme
nt 1/Part 1 - JJ/server.py"
Server listening on 127.0.0.1:12345
```

Step 3: Afterwards, start client.py in another terminal. It should be like this below.

```
(py312) joyanta@Joyantas-MacBook-Pro CS745 GA1 % python -u "/Users/joya
nta/UAB PhD/Spring 24/CS745/Assignment 1/Part 1 - JJ/client.py"
Connected to the server. You can start sending messages.
```

And the server.py should show something like this below.

```
Accepted connection from ('127.0.0.1', 55664)
```

Step 4: From client.py, you can start messaging, which is shown below.

```
hi
User A received: hello
how are you?
User A received: i am doing good.
```

From server.py, it should start receiving the message, like below.

```
User B received: hi
Enter your message: hello
User B received: how are you?
Enter your message: i am doing good.
```

And when it is being used in different machines, we can replace '0.0.0.0' with the server's IP address in client.py.

Part 2:

Open your terminal and go to the location where your client.py and server.py files are saved.

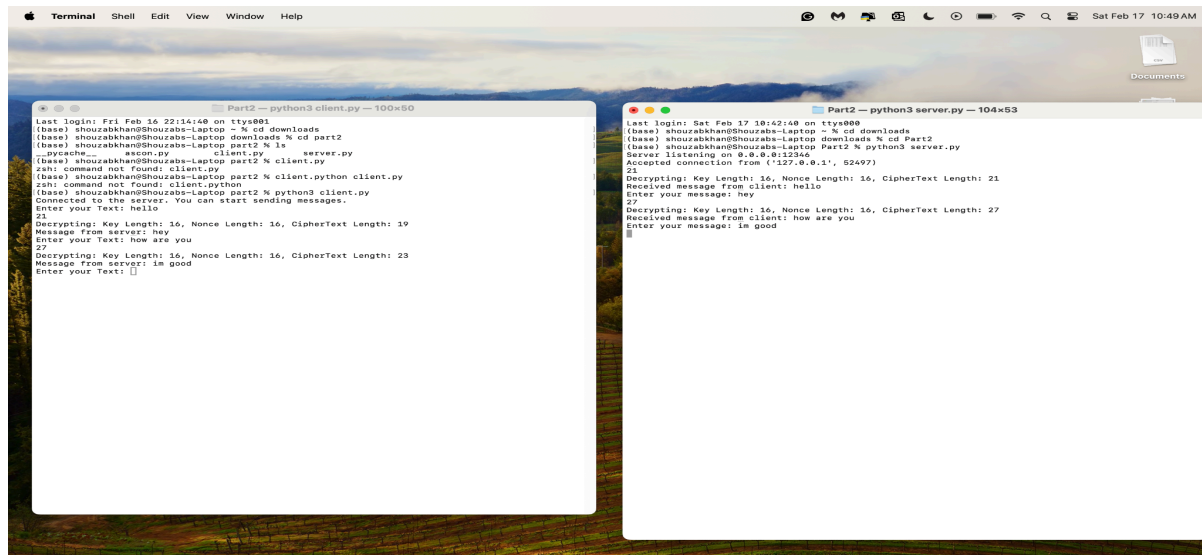
First run the server.py file through terminal “Python3 server.py” it will run the server file and will give you a message that the server is connected.

After that, open another terminal and run the client.py file “Python3 client.py” will run the client file and will give you a message that “Connected to server, You can send message” after that you can start sending messages which will be encrypted.

While running the code on different Systems you have to change the client.py and server.py code accordingly:

sock.bind(('0.0.0.0', 12342)) —> change 0.0.0.0 with the server's/Client's IP address.

Screenshots of real time test runs of your code:



The image shows two terminal windows side-by-side on a macOS desktop. The left window, titled 'Part2 - python3 client.py - 100x50', shows the client program's execution. It starts with a login timestamp, then prompts for a command. The user enters 'client.py', and the program connects to the server. It then receives a message 'hey' and sends back 'hello'. The right window, titled 'Part2 - python3 server.py - 104x53', shows the server program's execution. It starts with a login timestamp, then prompts for a command. The user enters 'server.py', and the program starts listening on 0.0.0.0:12342. It then receives a connection from 127.0.0.1:52497. It receives a message 'hello' and sends back 'hey'. It then receives a message 'how are you' and sends back 'im good'.

```
Part2 - python3 client.py - 100x50
Last login: Fri Feb 16 22:14:40 on ttys001
(base) shourabhkhan@Shourabs-Laptop ~ % cd downloads
(base) shourabhkhan@Shourabs-Laptop ~ % cd part2
(base) shourabhkhan@Shourabs-Laptop ~ % cd part2
(base) shourabhkhan@Shourabs-Laptop ~ % python3 client.py
zsh: command not found: client.py
(base) shourabhkhan@Shourabs-Laptop ~ % python3 client.py
zsh: command not found: client.py
(base) shourabhkhan@Shourabs-Laptop ~ % python3 client.py
Connected to the server. You can start sending messages.
Enter your Text: hello
23
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length: 19
Message from server: hey
Enter your Text: how are you
27
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length: 23
Message from server: im good
Enter your Text:

Part2 - python3 server.py - 104x53
Last login: Sat Feb 17 18:42:40 on ttys000
(base) shourabhkhan@Shourabs-Laptop ~ % cd downloads
(base) shourabhkhan@Shourabs-Laptop ~ % cd part2
(base) shourabhkhan@Shourabs-Laptop ~ % cd part2
(base) shourabhkhan@Shourabs-Laptop ~ % python3 server.py
Server listening on 0.0.0.0:12342
Accepted connection from ('127.0.0.1', 52497)
23
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length: 21
Received message from client: hello
Enter your message: hey
27
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length: 27
Received message from client: how are you
Enter your message: im good
28
```

Part 3:

Server Program: The server program is designed to listen for incoming connections on IP address '0.0.0.0' and port 12342. It accepts connections and waits for key, nonce, associated data, and ciphertext length. It receives ciphertext based on the length and attempts to decrypt it using AES-GCM. The server supports multiple messages from the client and continues to listen for new connections.

Client Program: The client program connects to the server's IP address '0.0.0.0' on port 12342. It generates a random nonce and AES-GCM key, sends them to the server along with associated data. The client then takes user input, encrypts the message using AES-GCM, and sends the ciphertext to the server.

1. Implemented a server that listens for incoming connections and handles multiple messages from clients.
2. Dynamically received key, nonce, associated data, and ciphertext length on both client and server.
3. Utilized the 'cryptography' library for AES-GCM encryption and decryption.

Steps to use this program:

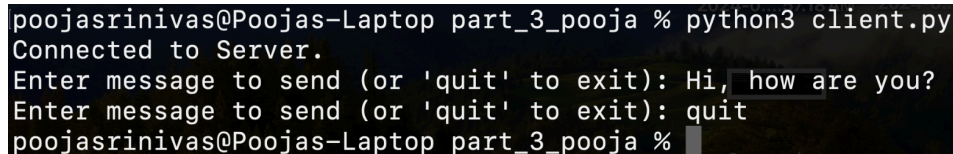
Step 1: Open two terminals (one for server another for client), change the directory to the existing file.

Step 2: Type “python3 server.py” in one terminal, then “python3 client.py” in another and press enter.

Step 3: Client gets connected to Server, type the message and press enter. Internally the Key, nonce, associated data is being shared to the server prior to sending a message. Message gets encrypted through AES.

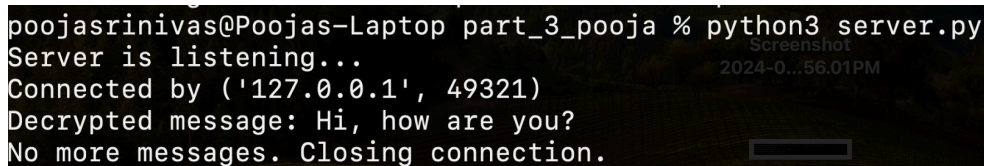
Step 4: Server receives the Key, nonce, associated data ahead of time and then once the server receives the message, it's been decrypted using the AES.

Step 5: If the client continues to send the message it gets delivered to the server else if the client quits the “No more messages. Closing connection msg is printed on the server terminal”

A screenshot of a terminal window showing the execution of client.py. The prompt is 'poojasrinivas@Poojas-Laptop part_3_pooja %'. The user enters 'python3 client.py', and the output is 'Connected to Server.'. Then, the user enters 'Hi, how are you?' and 'quit' in response to the prompts 'Enter message to send (or \'quit\' to exit):'.

```
poojasrinivas@Poojas-Laptop part_3_pooja % python3 client.py
Connected to Server.
Enter message to send (or 'quit' to exit): Hi, how are you?
Enter message to send (or 'quit' to exit): quit
poojasrinivas@Poojas-Laptop part_3_pooja %
```

client.py

A screenshot of a terminal window showing the execution of server.py. The prompt is 'poojasrinivas@Poojas-Laptop part_3_pooja %'. The user enters 'python3 server.py', and the output is 'Server is listening...'. Then, the user enters 'Hi, how are you?' and the output is 'Decrypted message: Hi, how are you?'. Finally, the user enters 'quit' and the output is 'No more messages. Closing connection.'. There are also several 'Screenshot' watermarks with timestamps from 2024-0...56.01PM to 2024-0...51.07PM.

```
poojasrinivas@Poojas-Laptop part_3_pooja % python3 server.py
Server is listening...
Connected by ('127.0.0.1', 49321)
Decrypted message: Hi, how are you?
No more messages. Closing connection.
```

server.py

Part 4:

Open your terminal and go to the location where your client.py and server.py files are saved. Make sure you have python 3 installed and use the pip install to download ascon library.

And if you are running both server.py and client.py in local system then make sure you change the

“def start_client(server_host='192.168.1.86', server_port=12346):”

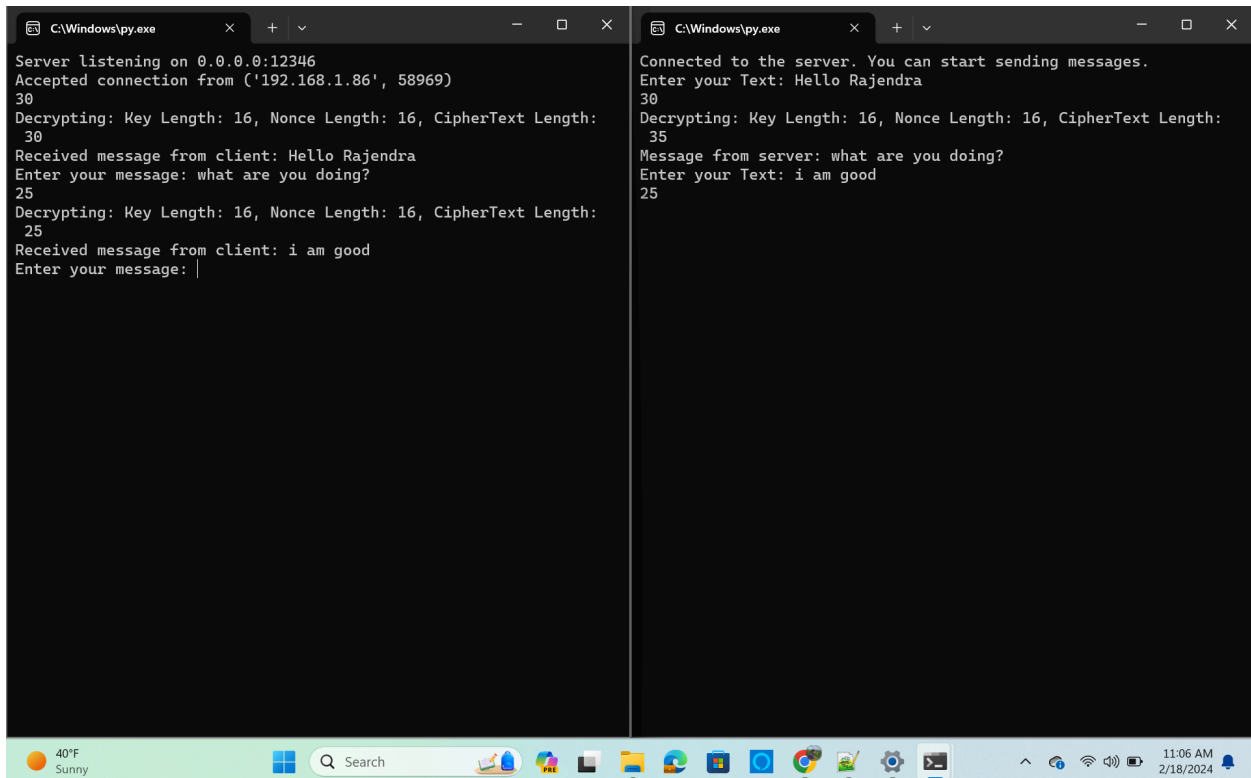
Server host ip address to ‘127.0.0.1’.

Else if you are running over the network make sure you will change the ip address to server.py running system ip address.

In my case, I am running over the network. First run the server.py file through terminal “Python3 server.py” it will run the server file and will give you a message that the server is connected. After that, open another terminal and run the client.py file “Python3 client.py” will run the client file and will give you a message that “Connected to server, You can send message” after that you can start sending messages which will be encrypted.

1. ascon.py: This script implements the Ascon v1.2 cipher, including its hash function, message authentication code (MAC), and AEAD (Authenticated Encryption with Associated Data) encryption and decryption. Here, two functions ascon_encrypt and ascon_decrypt pass a parameter for AEAD which is “variant=“Ascon-128””. It also includes various helper functions for byte manipulation and debugging.
2. client.py: This script implements the client-side functionality for the secure messaging system. It includes functions for encrypting and decrypting text messages using Ascon (ascon-128), as well as functions for communicating with the server using sockets.
3. server.py: This script implements the server-side functionality for the secure messaging system. It includes functions for encrypting and decrypting text messages using Ascon (ascon-128), as well as functions for communicating with clients using sockets.

Screenshots:



```
C:\Windows\py.exe x + - □ x
Server listening on 0.0.0.0:12346
Accepted connection from ('192.168.1.86', 58969)
30
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length:
30
Received message from client: Hello Rajendra
Enter your message: what are you doing?
25
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length:
25
Received message from client: i am good
Enter your message: |

C:\Windows\py.exe x + - □ x
Connected to the server. You can start sending messages.
Enter your Text: Hello Rajendra
30
Decrypting: Key Length: 16, Nonce Length: 16, CipherText Length:
35
Message from server: what are you doing?
Enter your Text: i am good
25
```

Every time I send a message from client to server and server to client, encryption and decryption is working on both sides. So server and client both are having two keys every time when they are getting connected.

Lesson Learned:

During implementing this software we as a group learned a lot of things:

- Learnt Socket Programming through which we connected to computers with each other for secure text exchanging.
- Importance of handling dynamic data lengths, especially when dealing with variable-sized messages.
- Ensured that the data sent by the client aligns with the expectations of the server for successful decryption.
- Learned about ascon cipher and pyascon packages for encryption and decryption.
- Learned about how to securely share the private key and associated data with the server for communication.

Future Enhancement:

- Implementation of logging for better tracking of events and potential issues during execution.
- We can use MAC addresses instead of IP addresses to have message integrity. So the message cannot be tampered.
- We can build a good user interface (UI) for transferring secure messages from client to server end.
- We can also upgrade from sending secure messages to sending secure file transfer.