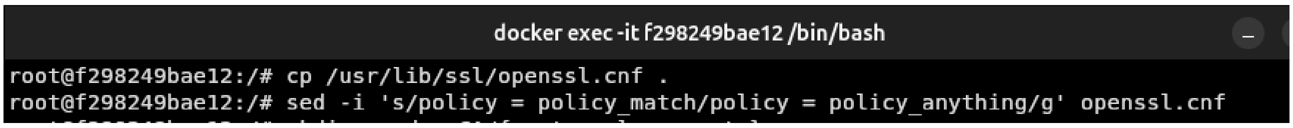
LAB 6

Part I: Create Server Public-Key and Certificate

Step 1: Setup OpenSSL Configuration

# Copy OpenSSL configuration file

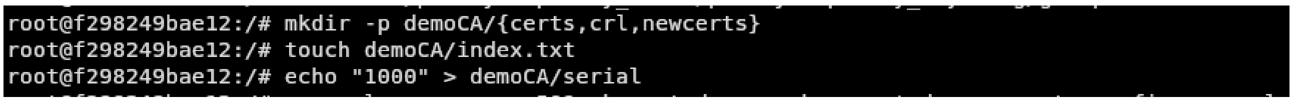
# Edit openssl.cnf file # Change "policy = policy_match" to "policy = policy_anything"



```
root@f298249bae12:/# cp /usr/lib/ssl/openssl.cnf .
root@f298249bae12:/# sed -i 's/policy = policy_match/policy = policy_anything/g' openssl.cnf
```

Step 2: Create Certificate Authority Structure

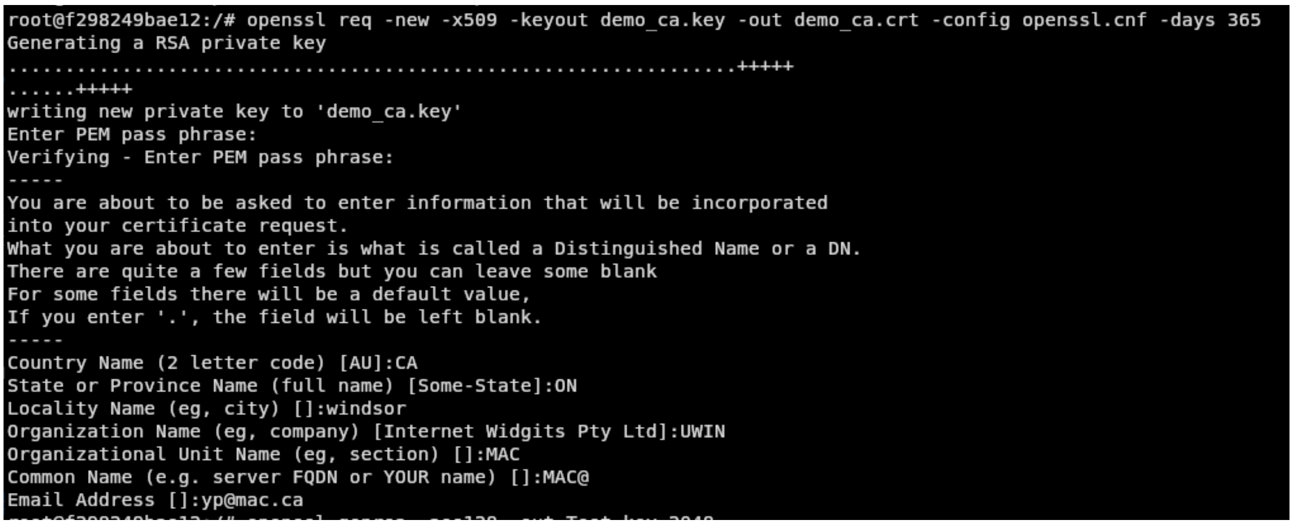
# Create demoCA directory structure



```
root@f298249bae12:/# mkdir -p demoCA/{certs,crl,newcerts}
root@f298249bae12:/# touch demoCA/index.txt
root@f298249bae12:/# echo "1000" > demoCA/serial
```

Step 3: Generate Certificate Authority (CA)

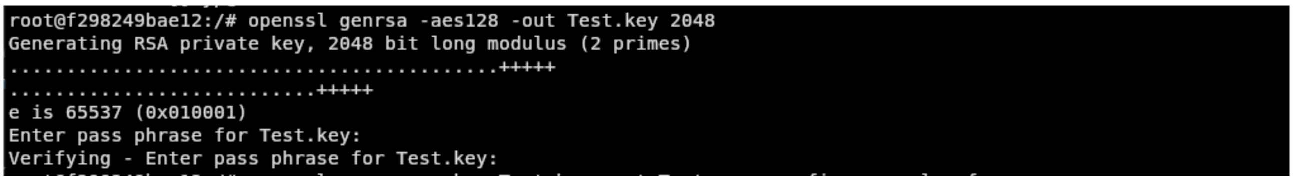
# Generate self-signed certificate for CA



```
root@f298249bae12:/# openssl req -new -x509 -keyout demo_ca.key -out demo_ca.crt -config openssl.cnf -days 365
Generating a RSA private key
..........................................................+++++
......+++++
writing new private key to 'demo_ca.key'
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CA
State or Province Name (full name) [Some-State]:ON
Locality Name (eg, city) []:windsor
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UWIN
Organizational Unit Name (eg, section) []:MAC
Common Name (e.g. server FQDN or YOUR name) []:MAC@
Email Address []:yp@mac.ca
```

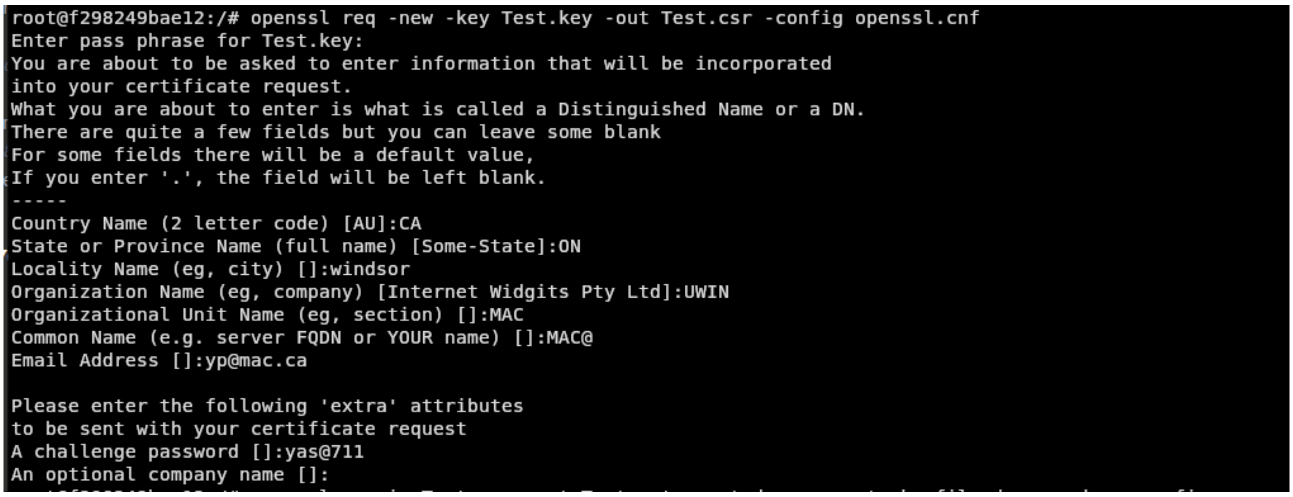
Step 4: Create TLS Server Certificate

# 1. Generate RSA private key for TLS server



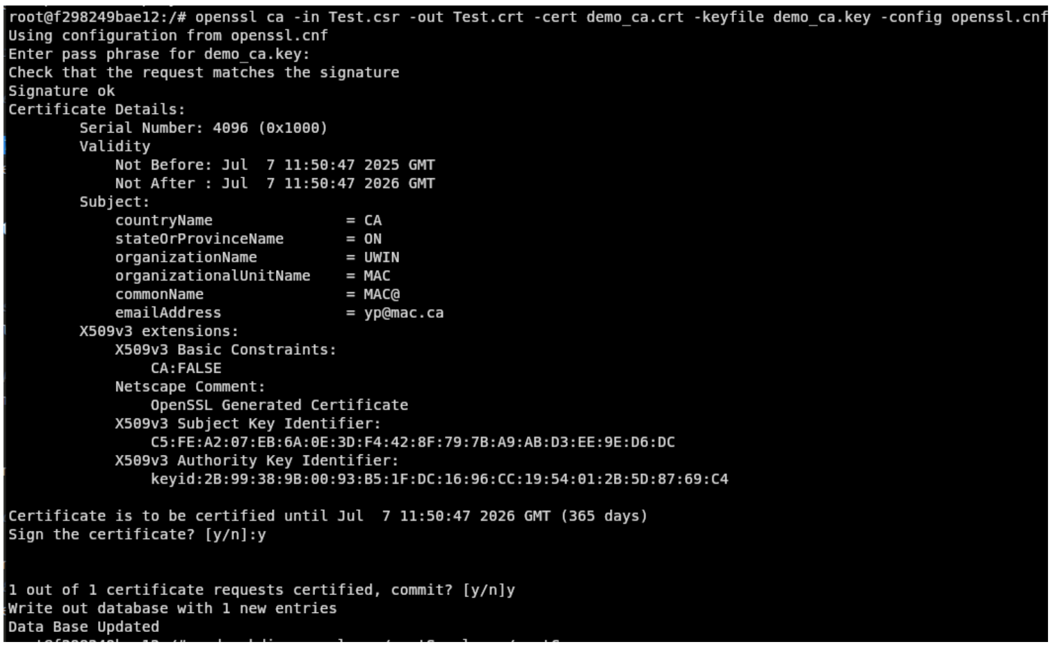
```
root@f298249bae12:/# openssl genrsa -aes128 -out Test.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
..........................................+++++
..........................+++++
e is 65537 (0x010001)
Enter pass phrase for Test.key:
Verifying - Enter pass phrase for Test.key:
```

# 2. Generate certificate signing request



```
root@f298249bae12:/# openssl req -new -key Test.key -out Test.csr -config openssl.cnf
Enter pass phrase for Test.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:CA
State or Province Name (full name) [Some-State]:ON
Locality Name (eg, city) []:windsor
Organization Name (eg, company) [Internet Widgits Pty Ltd]:UWIN
Organizational Unit Name (eg, section) []:MAC
Common Name (e.g. server FQDN or YOUR name) []:MAC@
Email Address []:yp@mac.ca

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:yas@711
An optional company name []:
```

# 3. Generate certificate for TLS server



```
root@f298249bae12:/# openssl ca -in Test.csr -out Test.crt -cert demo_ca.crt -keyfile demo_ca.key -config openssl.cnf
Using configuration from openssl.cnf
Enter pass phrase for demo_ca.key:
Check that the request matches the signature
Signature ok
Certificate Details:
        Serial Number: 4096 (0x1000)
        Validity
            Not Before: Jul  7 11:50:47 2025 GMT
            Not After : Jul  7 11:50:47 2026 GMT
        Subject:
            countryName               = CA
            stateOrProvinceName       = ON
            organizationName          = UWIN
            organizationalUnitName    = MAC
            commonName                = MAC@
            emailAddress              = yp@mac.ca
        X509v3 extensions:
            X509v3 Basic Constraints:
                CA:FALSE
            Netscape Comment:
                OpenSSL Generated Certificate
            X509v3 Subject Key Identifier:
                C5:FE:A2:07:EB:6A:0E:3D:F4:42:8F:79:7B:A9:AB:D3:EE:9E:D6:DC
            X509v3 Authority Key Identifier:
                keyid:2B:99:38:9B:00:93:B5:1F:DC:16:96:CC:19:54:01:2B:5D:87:69:C4

Certificate is to be certified until Jul  7 11:50:47 2026 GMT (365 days)
Sign the certificate? [y/n]:y


1 out of 1 certificate requests certified, commit? [y/n]y
Write out database with 1 new entries
Data Base Updated
```

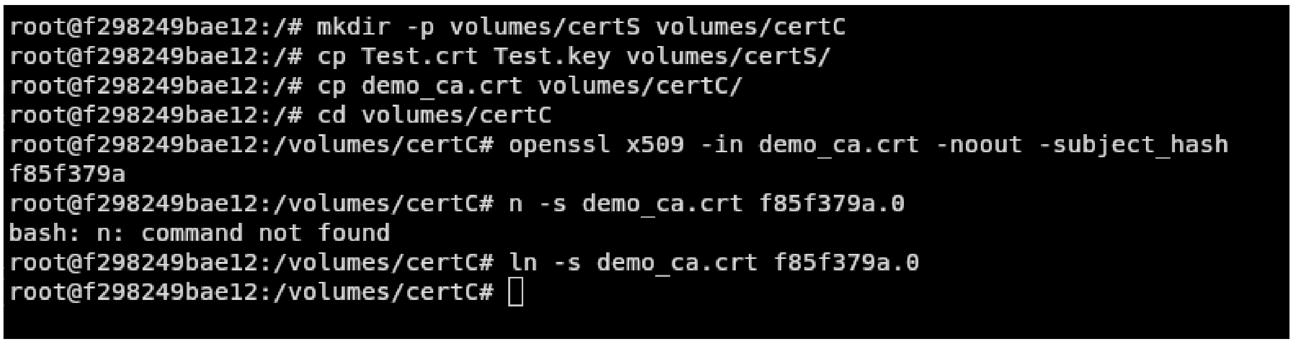
Step 5: Setup Certificate Directories

# Create directories in volumes folder

# Copy server certificate and key

# Copy CA certificate

# Create symbolic link for CA certificate



```
root@f298249bae12:/# mkdir -p volumes/certS volumes/certC
root@f298249bae12:/# cp Test.crt Test.key volumes/certS/
root@f298249bae12:/# cp demo_ca.crt volumes/certC/
root@f298249bae12:/# cd volumes/certC
root@f298249bae12:/volumes/certC# openssl x509 -in demo_ca.crt -noout -subject_hash
f85f379a
root@f298249bae12:/volumes/certC# n -s demo_ca.crt f85f379a.0
bash: n: command not found
root@f298249bae12:/volumes/certC# ln -s demo_ca.crt f85f379a.0
root@f298249bae12:/volumes/certC# 
```

Part II: TLS Client and Server Implementation

TLS Server Code (server.py)

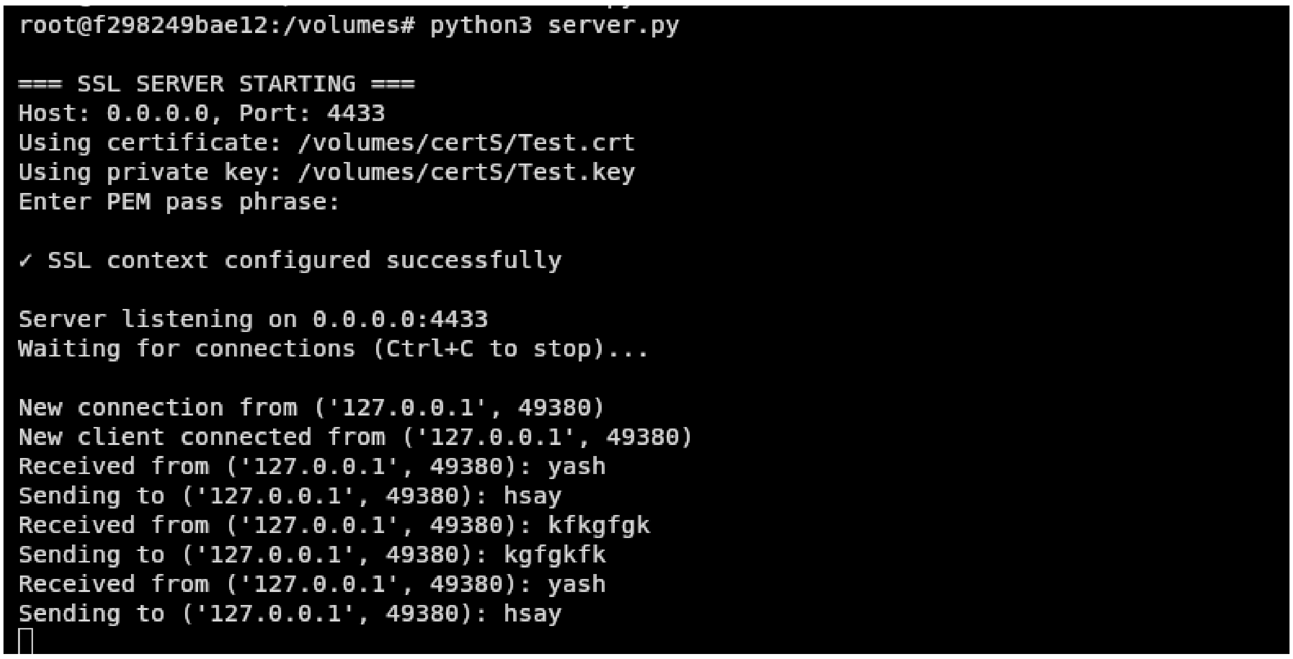
```
#!/usr/bin/env python3

import socket
import ssl
import threading
import sys

def handle_client(conn, addr):
    """Handle individual client connections"""
    print(f"New client connected from {addr}")

    try:
        while True:
            # Receive message from client
            data = conn.recv(1024)
            if not data:
                break

            message = data.decode('utf-8').strip()
            print(f"Received from {addr}: {message}")

            # Reverse the message
            reversed_message = message[::-1]
            print(f"Sending to {addr}: {reversed_message}")
            # Send reversed message back
            conn.send(reversed_message.encode('utf-8'))

    except Exception as e:
        print(f"Error handling client {addr}: {e}")
    finally:
        conn.close()
        print(f"Client {addr} disconnected")

def main():
    # Server configuration
    HOST = '0.0.0.0'
    PORT = 4433

    # Certificate paths
    certfile = 'volumes/certS/Test.crt'
    keyfile = 'volumes/certS/Test.key'

    # Create SSL context
    context = ssl.create_default_context(ssl.Purpose.CLIENT_AUTH)
    context.load_cert_chain(certfile, keyfile)

    # Create socket
    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, PORT))
    server_socket.listen(5)

    print(f"TLS Server listening on {HOST}:{PORT}")

    try:
        while True:
            # Accept new connection
            client_socket, addr = server_socket.accept()

            # Wrap socket with SSL
            ssl_client_socket = context.wrap_socket(client_socket, server_side=True)

            # Create thread for each client
            client_thread = threading.Thread(
                target=handle_client,
                args=(ssl_client_socket, addr)
            )
            client_thread.daemon = True
            client_thread.start()

    except KeyboardInterrupt:
        print("\nShutting down server...")
    finally:
        server_socket.close()

if __name__ == "__main__":
    main()
```

TLS Client Code (client.py)

```
#!/usr/bin/env python3

import socket
import ssl
import sys

def main():
    if len(sys.argv) != 2:
        print("Usage: python3 client.py <server_hostname>")
        sys.exit(1)

    server_hostname = sys.argv[1]
    server_port = 4433

    # CA certificate path
    ca_cert_path = 'volumes/certC'

    # Create SSL context
    context = ssl.create_default_context()
    context.check_hostname = True
    context.verify_mode = ssl.CERT_REQUIRED
    context.load_verify_locations(capath=ca_cert_path)

    try:
        # Create socket and connect
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        # Wrap socket with SSL
        ssl_sock = context.wrap_socket(
            sock,
            server_hostname=server_hostname
        )

        # Connect to server
        ssl_sock.connect((server_hostname, server_port))

        print(f"Connected to {server_hostname}:{server_port}")
        print("TLS connection established!")
        print("Certificate verification successful!")
        print("\nEnter messages to send to server (type 'quit' to exit):")

        while True:
            # Get user input
            message = input("Enter message: ").strip()

            if message.lower() == 'quit':
                break
            if message:
                # Send message to server
                ssl_sock.send(message.encode('utf-8'))

                # Receive response
                response = ssl_sock.recv(1024)
                reversed_message = response.decode('utf-8')

                print(f"Server response: {reversed_message}")

    except ssl.SSLError as e:
        print(f"SSL Error: {e}")
    except ConnectionRefusedError:
        print("Connection refused. Make sure server is running.")
    except Exception as e:
        print(f"Error: {e}")
    finally:
        try:
            ssl_sock.close()
        except:
            pass

if __name__ == "__main__":
    main()
```

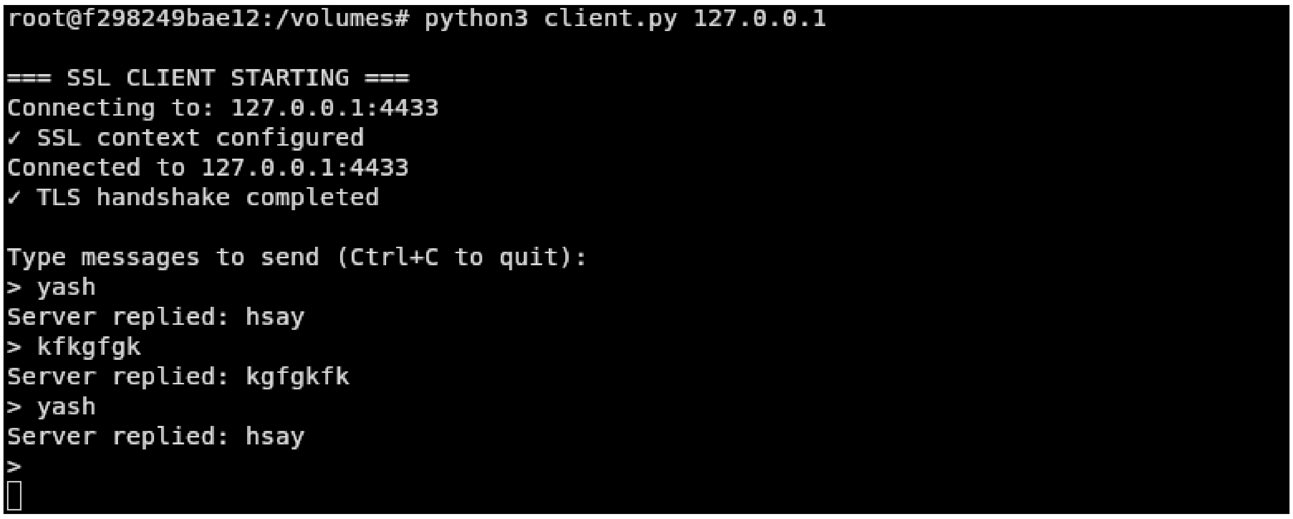
Output:

server side:



```
root@f298249bae12:/volumes# python3 server.py

=== SSL SERVER STARTING ===
Host: 0.0.0.0, Port: 4433
Using certificate: /volumes/certS/Test.crt
Using private key: /volumes/certS/Test.key
Enter PEM pass phrase:

✓ SSL context configured successfully

Server listening on 0.0.0.0:4433
Waiting for connections (Ctrl+C to stop)...

New connection from ('127.0.0.1', 49380)
New client connected from ('127.0.0.1', 49380)
Received from ('127.0.0.1', 49380): yash
Sending to ('127.0.0.1', 49380): hsay
Received from ('127.0.0.1', 49380): kfkgfgk
Sending to ('127.0.0.1', 49380): kgfgkfk
Received from ('127.0.0.1', 49380): yash
Sending to ('127.0.0.1', 49380): hsay
```

Client side:



```
root@f298249bae12:/volumes# python3 client.py 127.0.0.1

=== SSL CLIENT STARTING ===
Connecting to: 127.0.0.1:4433
✓ SSL context configured
Connected to 127.0.0.1:4433
✓ TLS handshake completed

Type messages to send (Ctrl+C to quit):
> yash
Server replied: hsay
> kfkgfgk
Server replied: kgfgkfk
> yash
Server replied: hsay
>
```