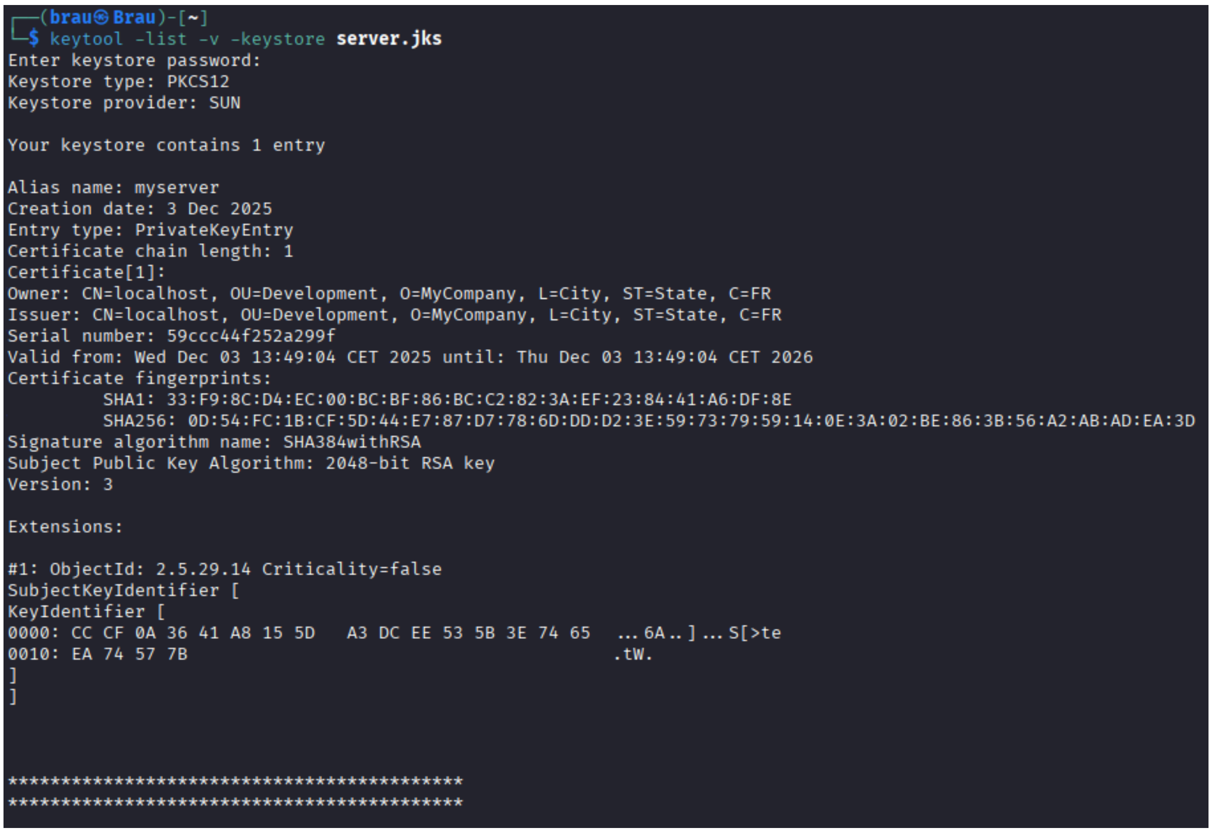
Tristan BRAU
Léo PHAV

Java pour les réseaux - Lab 5

2 - Practical Exercises

2.1 - Exercise 1: Setting Up SSL/TLS Environment

Verification of the key creation:



```
┌──(brau㉿Brau)-[~]
└─$ keytool -list -v -keystore server.jks
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: myserver
Creation date: 3 Dec 2025
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=localhost, OU=Development, O=MyCompany, L=City, ST=State, C=FR
Issuer: CN=localhost, OU=Development, O=MyCompany, L=City, ST=State, C=FR
Serial number: 59ccc44f252a299f
Valid from: Wed Dec 03 13:49:04 CET 2025 until: Thu Dec 03 13:49:04 CET 2026
Certificate fingerprints:
         SHA1: 33:F9:8C:D4:EC:00:BC:BF:86:BC:C2:82:3A:EF:23:84:41:A6:DF:8E
         SHA256: 0D:54:FC:1B:CF:5D:44:E7:87:D7:78:6D:DD:D2:3E:59:73:79:59:14:0E:3A:02:BE:86:3B:56:A2:AB:AD:EA:3D
Signature algorithm name: SHA384withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: CC CF 0A 36 41 A8 15 5D   A3 DC EE 53 5B 3E 74 65  ...6A..]...S[>te
0010: EA 74 57 7B                                        .tW.
]
]



*******************************************
*******************************************
```

2.1.2 - Task 1.2: Understand Certificate Contents

Analysis Questions:

- What is the certificate's owner name?

Certificate Owner Name:
CN=localhost, OU=Development, O=MyCompany, L=City, ST=State, C=FR

- Who issued this certificate?

Issuer of the Certificate:
CN=localhost, OU=Development, O=MyCompany, L=City, ST=State, C=FR

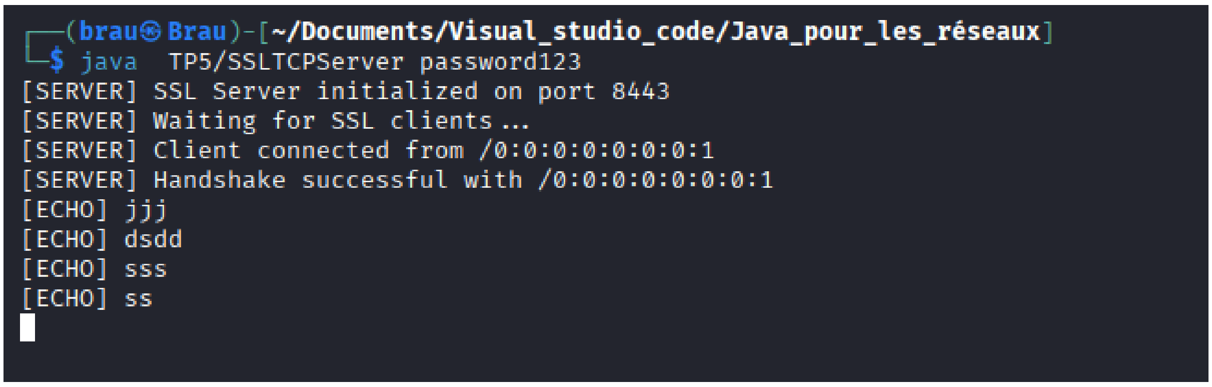
- What encryption algorithm was used?

Encryption Algorithm Used:
SHA384withRSA (Signature algorithm)
Subject Public Key Algorithm: 2048-bit RSA key

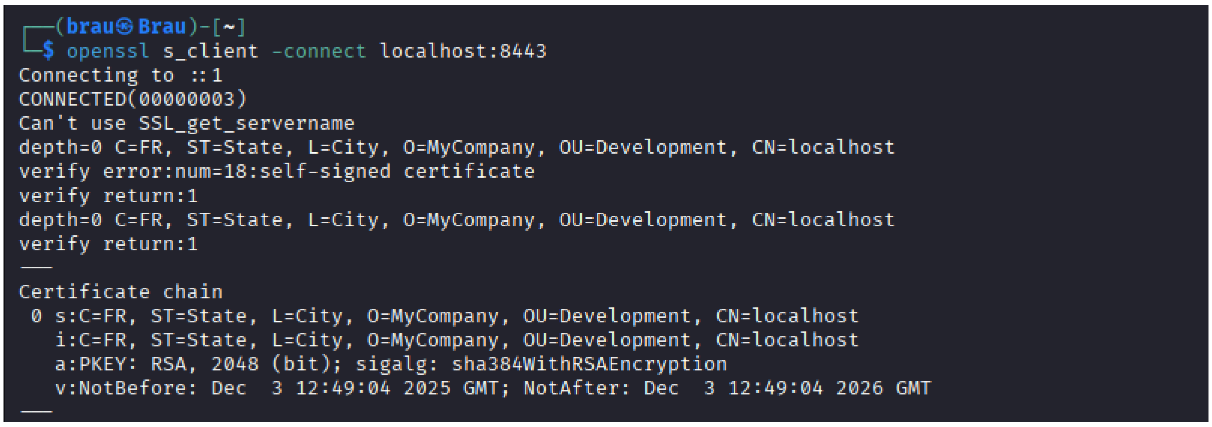
- When does the certificate expire?

Certificate Expiration Date:
Thu Dec 03 13:49:04 CET 2026

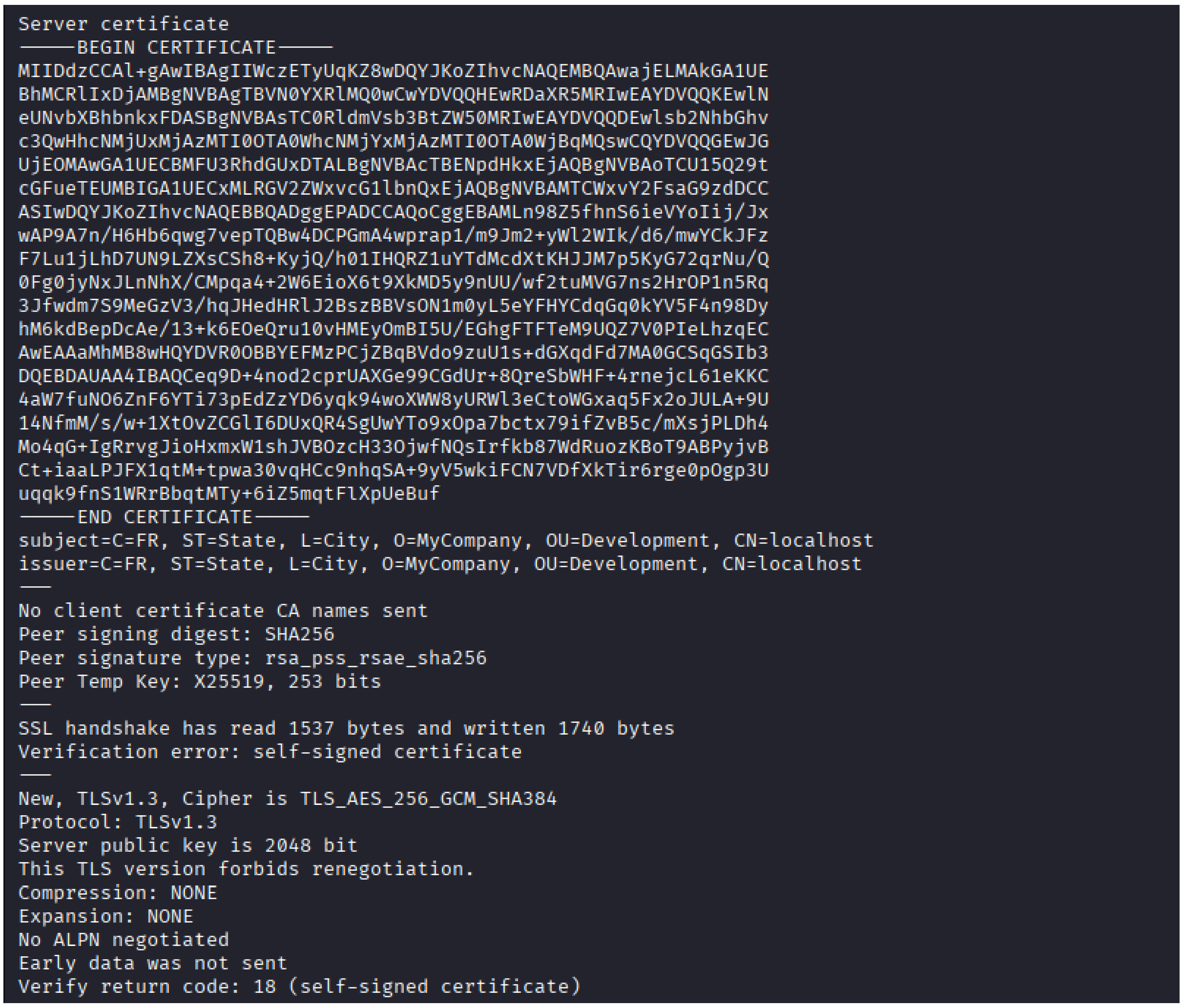
2.2 - Exercise 2: SSL Server Implementation



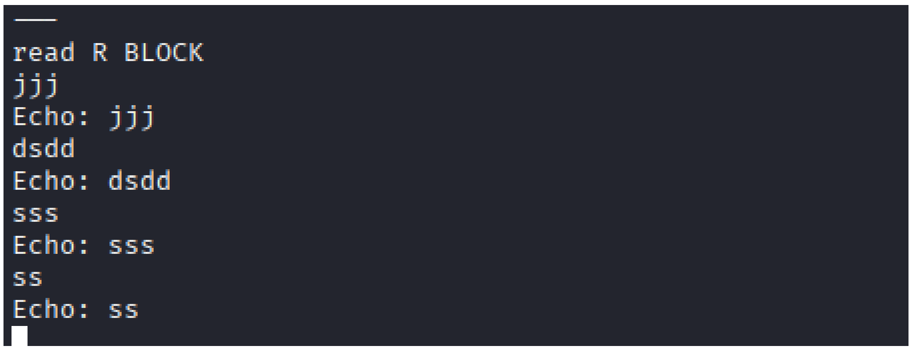
```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLTCPServer password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[SERVER] Handshake successful with /0:0:0:0:0:0:0:1
[ECHO] jjj
[ECHO] dsdd
[ECHO] sss
[ECHO] ss
```



```
┌──(brau㉿Brau)-[~]
└─$ openssl s_client -connect localhost:8443
Connecting to ::1
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
verify error:num=18:self-signed certificate
verify return:1
depth=0 C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
verify return:1
---
Certificate chain
 0 s:C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
   i:C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
   a:PKEY: RSA, 2048 (bit); sigalg: sha384WithRSAEncryption
   v:NotBefore: Dec  3 12:49:04 2025 GMT; NotAfter: Dec  3 12:49:04 2026 GMT
---
```



```
Server certificate
-----BEGIN CERTIFICATE-----
MIIDdzCCAl+gAwIBAgIIWczETyUqKZ8wDQYJKoZIhvcNAQEMBQAwajELMAkGA1UE
BhMCRlIxDjAMBgNVBAgTBVN0YXRlMQ0wCwYDVQQHEwRDaXR5MRIwEAYDVQQKEwlN
eUNvbXBhbnkxFDASBgNVBAsTC0RldmVsb3BtZW50MRIwEAYDVQQDEwlsb2NhbGhv
c3QwHhcNMjUxMjAzMTI0OTA0WhcNMjYxMjAzMTI0OTA0WjBqMQswCQYDVQQGEwJG
UjEOMAwGA1UECBMFU3RhdGUxDTALBgNVBAcTBENpdHkxEjAQBgNVBAoTCU15Q29t
cGFueTEUMBIGA1UECxMLRGV2ZWxvcG1lbnQxEjAQBgNVBAMTCWxvY2FsaG9zdDCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMLn98Z5fhnS6ieVYoIij/Jx
wAP9A7n/H6Hb6qwg7vepTQBw4DCPGmA4wprap1/m9Jm2+yWl2WIk/d6/mwYCkJFz
F7Lu1jLhD7UN9LZXsCSh8+KyjQ/h01IHQRZ1uYTdMcdXtKHJJM7p5KyG72qrNu/Q
0Fg0jyNxJLnNhX/CMpqa4+2W6EioX6t9XkMD5y9nUU/wf2tuMVG7ns2HrOP1n5Rq
3Jfwdm7S9MeGzV3/hqJHedHRlJ2BszBBVsON1m0yL5eYFHYCdqGq0kYV5F4n98Dy
hM6kdBepDcAe/13+k6EOeQru10vHMEyOmBI5U/EGhgFTFTeM9UQZ7V0PIeLhzqEC
AwEAAaMhMB8wHQYDVR0OBBYEFMzPCjZBqBVdo9zuU1s+dGXqdFd7MA0GCSqGSIb3
DQEBDAUAA4IBAQCeq9D+4nod2cprUAXGe99CGdUr+8QreSbWHF+4rnejcL61eKKC
4aW7fuNO6ZnF6YTi73pEdZzYD6yqk94woXWW8yURWl3eCtoWGxaq5Fx2oJULA+9U
14NfmM/s/w+1XtOvZCGlI6DUxQR4SgUwYTo9xOpa7bctx79ifZvB5c/mXsjPLDh4
Mo4qG+IgRrvgJioHxmxW1shJVBOzcH33OjwfNQsIrfkb87WdRuozKBoT9ABPyjvB
Ct+iaaLPJFX1qtM+tpwa30vqHCc9nhqSA+9yV5wkiFCN7VDfXkTir6rge0pOgp3U
uqqk9fnS1WRrBbqtMTy+6iZ5mqtFlXpUeBuf
-----END CERTIFICATE-----
subject=C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
issuer=C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: rsa_pss_rsae_sha256
Peer Temp Key: X25519, 253 bits
---
SSL handshake has read 1537 bytes and written 1740 bytes
Verification error: self-signed certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Protocol: TLSv1.3
Server public key is 2048 bit
This TLS version forbids renegotiation.
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self-signed certificate)
```



```
---
read R BLOCK
jjj
Echo: jjj
dsdd
Echo: dsdd
sss
Echo: sss
ss
Echo: ss
```

2.2.4 Testing Your Implementation

To test our SSL server implementation, we first launched our server on port 8443, then we used OpenSSL to verify the TLS handshake, certificate exchange, and encrypted communication.

Starting the SSL Server

We started the server using the command:
java TP5/SSLTCPServer password123

The server console confirmed that the SSL context and keystore were correctly loaded, and that the server was listening on port 8443.

Testing with OpenSSL

We connected to the server using:
openssl s_client -connect localhost:8443

OpenSSL successfully established a secure TLS session with our server. The output showed:

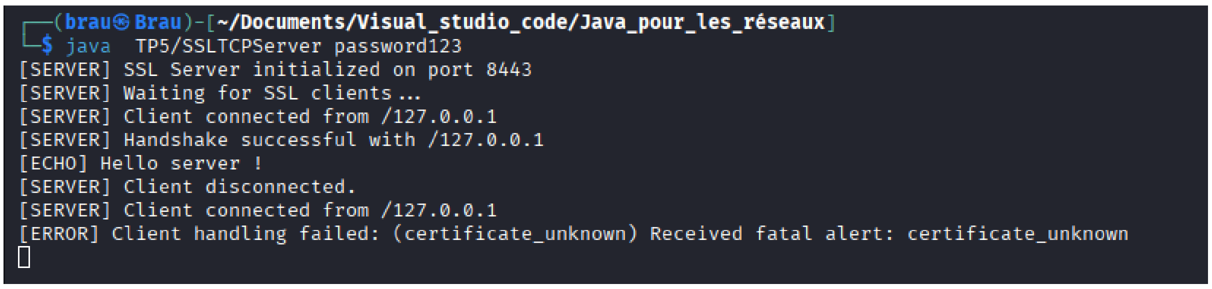
- TLS version: **TLSv1.3**
- Cipher suite: **TLS_AES_256_GCM_SHA384**
- The full server certificate, matching the information we provided during keystore creation (CN=localhost, OU=Development, O=MyCompany…)
- The expected verification warning: *“self-signed certificate”*

Since our certificate is self-signed for local testing, this warning is normal.

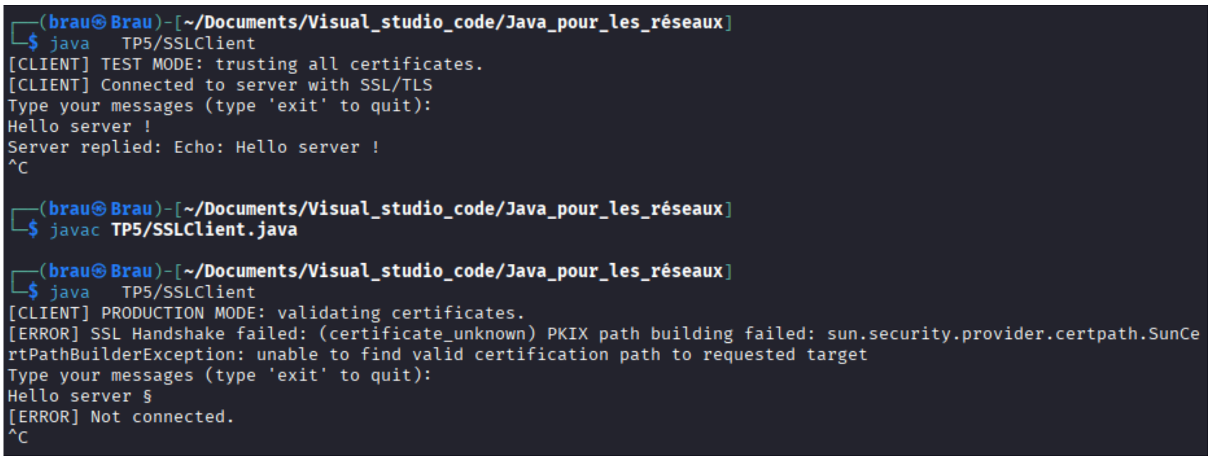
Echo Functionality Test

After the handshake, we sent several test messages through OpenSSL. For each message, the server correctly replied by echoing the content.

2.3 - Exercise 3: SSL Client Implementation



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLTCPServer password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /127.0.0.1
[SERVER] Handshake successful with /127.0.0.1
[ECHO] Hello server !
[SERVER] Client disconnected.
[SERVER] Client connected from /127.0.0.1
[ERROR] Client handling failed: (certificate_unknown) Received fatal alert: certificate_unknown
```



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLClient
[CLIENT] TEST MODE: trusting all certificates.
[CLIENT] Connected to server with SSL/TLS
Type your messages (type 'exit' to quit):
Hello server !
Server replied: Echo: Hello server !
^C

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ javac TP5/SSLClient.java

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLClient
[CLIENT] PRODUCTION MODE: validating certificates.
[ERROR] SSL Handshake failed: (certificate_unknown) PKIX path building failed: sun.security.provider.certpath.SunCe
rtPathBuilderException: unable to find valid certification path to requested target
Type your messages (type 'exit' to quit):
Hello server §
[ERROR] Not connected.
^C
```

To test our SSL client implementation, we ran the client in both testing mode (trust all certificates) and production mode (validate certificates).

Client Behavior in Testing Mode

We first launched the server, then started the client in test mode, which trusts all certificates without performing validation.
After the connection was established, we were able to send messages. This confirms that in testing mode:

- The handshake completes successfully.
- The client accepts the server's self-signed certificate.
- Encrypted communication works fully.

Certificate Validation Errors in Production Mode

We then recompiled and relaunched the client in production mode, where certificates must be properly validated against a trust store.
In this mode, the client failed to validate the server certificate and showed the following error:
SSL Handshake failed: (certificate_unknown)
PKIX path building failed: unable to find valid certification path to requested target

At the same time, the server console showed:
ERROR: Received fatal alert: certificate_unknown

This behavior is expected because:

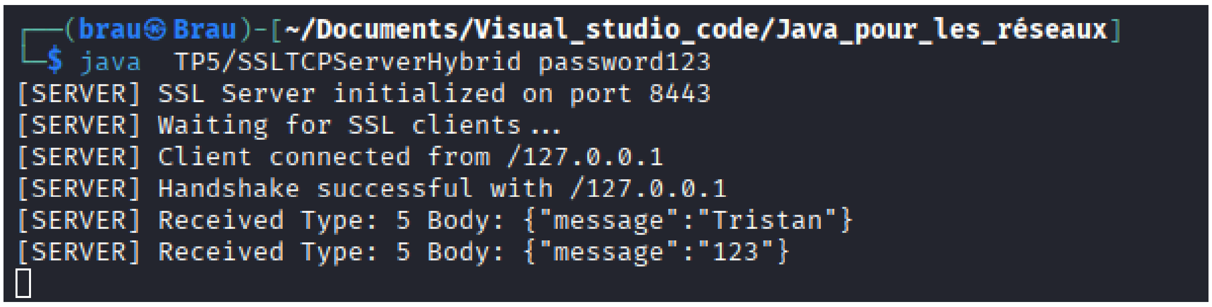
- Our server uses a self-signed certificate, not trusted by the default JVM trust store.
- In production mode, the client requires a certificate signed by a trusted Certificate Authority (CA), or the server's certificate must be manually imported into the client's trust store.

Since the certificate cannot be verified, the handshake fails and the client is not able to send messages securely.

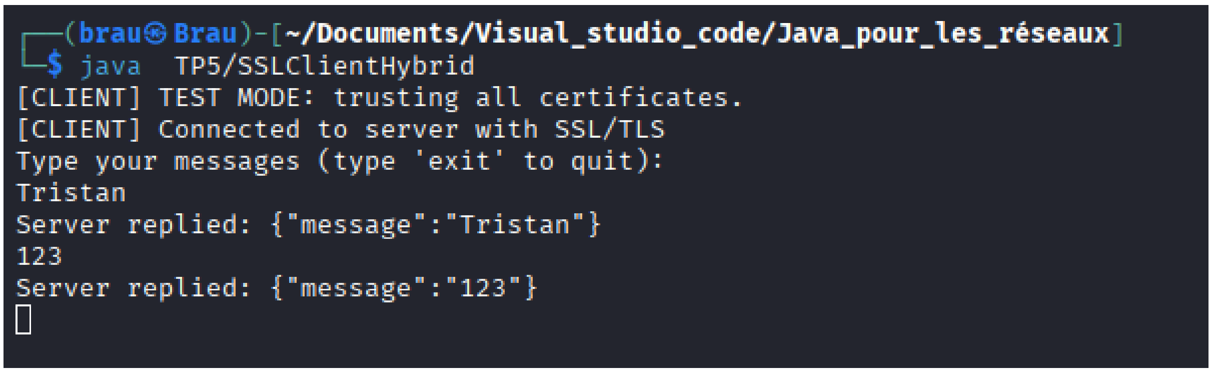
Tristan BRAU
Léo PHAV

3 - Custom Protocol Design

3.1 - Exercise 4: Chat Protocol Specification



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLTCPServerHybrid password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /127.0.0.1
[SERVER] Handshake successful with /127.0.0.1
[SERVER] Received Type: 5 Body: {"message":"Tristan"}
[SERVER] Received Type: 5 Body: {"message":"123"}
```



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLClientHybrid
[CLIENT] TEST MODE: trusting all certificates.
[CLIENT] Connected to server with SSL/TLS
Type your messages (type 'exit' to quit):
Tristan
Server replied: {"message":"Tristan"}
123
Server replied: {"message":"123"}
```

For this exercise, we implemented a simple custom chat protocol on top of our SSL/TLS connection. The goal was to send structured messages instead of plain text, using a combination of a message type and a JSON body. Our protocol encoder/decoder was tested using the hybrid server and client.

Server Reception of Protocol Messages

We first launched the hybrid SSL server:
java TP5/SSLTCPServerHybrid password123

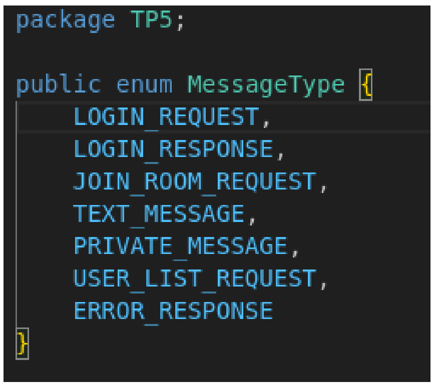
The server started correctly, initialized the SSL context, and waited for incoming SSL clients. After the client connected, the server successfully completed the TLS handshake and began receiving protocol messages.
The server correctly extracted the message type and correctly parsed the JSON message body.

Client Sending Protocol Messages

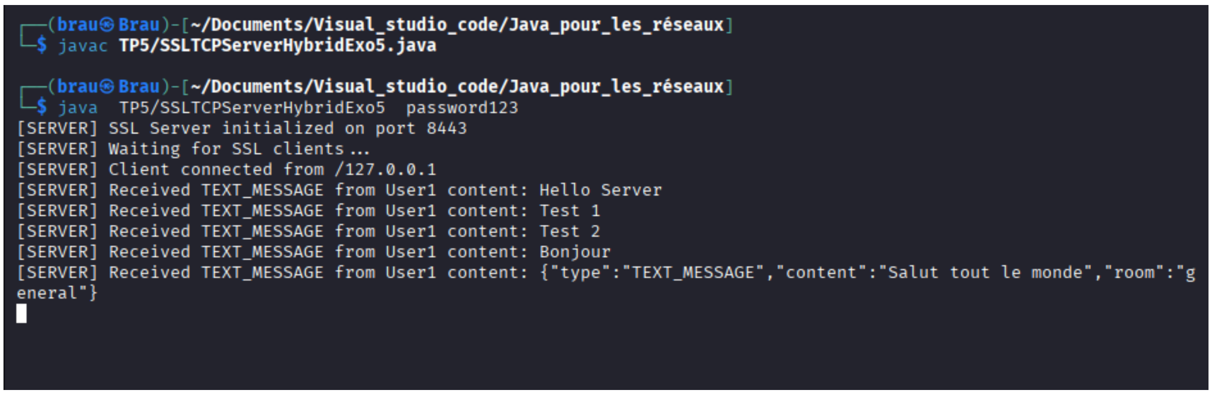
On the client side, we launched:
java TP5/SSLClientHybrid

The client ran in test mode and successfully established an SSL/TLS connection with the server. Once connected, we typed several messages and each input was encoded into our protocol format: a message type (Type 5) and a JSON body containing the user's text. The server processed and echoed back structured protocol responses.

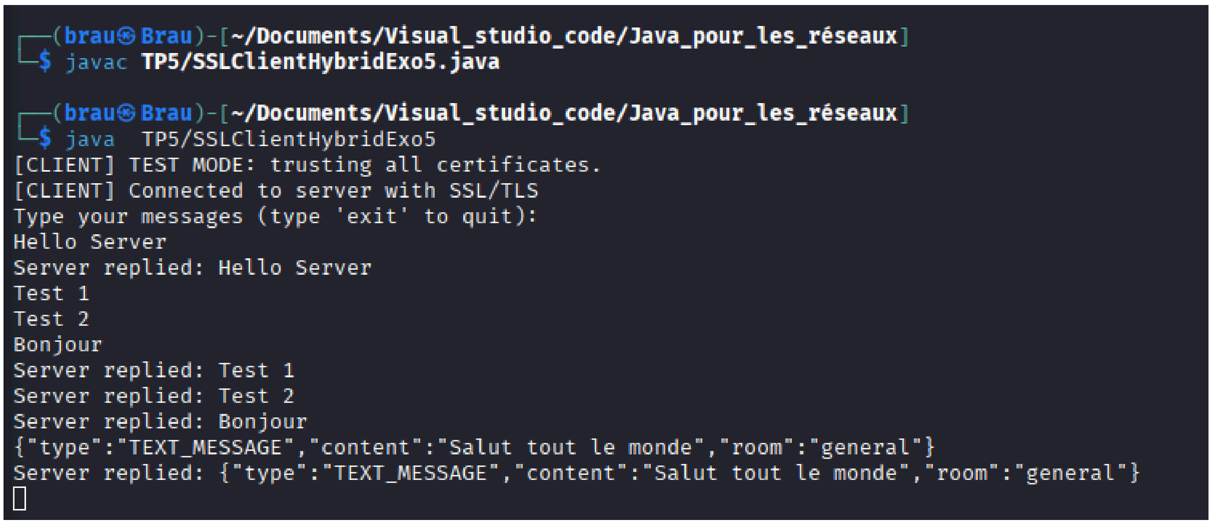
3.2 - Exercise 5: Protocol Message Design



```
package TP5;

public enum MessageType {
    LOGIN_REQUEST,
    LOGIN_RESPONSE,
    JOIN_ROOM_REQUEST,
    TEXT_MESSAGE,
    PRIVATE_MESSAGE,
    USER_LIST_REQUEST,
    ERROR_RESPONSE
}
```



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ javac TP5/SSLTCPServerHybridExo5.java

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients ...
[SERVER] Client connected from /127.0.0.1
[SERVER] Received TEXT_MESSAGE from User1 content: Hello Server
[SERVER] Received TEXT_MESSAGE from User1 content: Test 1
[SERVER] Received TEXT_MESSAGE from User1 content: Test 2
[SERVER] Received TEXT_MESSAGE from User1 content: Bonjour
[SERVER] Received TEXT_MESSAGE from User1 content: {"type":"TEXT_MESSAGE","content":"Salut tout le monde","room":"g
eneral"}
```



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ javac TP5/SSLClientHybridExo5.java

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java  TP5/SSLClientHybridExo5
[CLIENT] TEST MODE: trusting all certificates.
[CLIENT] Connected to server with SSL/TLS
Type your messages (type 'exit' to quit):
Hello Server
Server replied: Hello Server
Test 1
Test 2
Bonjour
Server replied: Test 1
Server replied: Test 2
Server replied: Bonjour
{"type":"TEXT_MESSAGE","content":"Salut tout le monde","room":"general"}
Server replied: {"type":"TEXT_MESSAGE","content":"Salut tout le monde","room":"general"}
```

We created the *MessageType* enum with all required protocol types. Then we tested our protocol by sending different text messages from the client.

The server correctly received and parsed each message. It displayed the type (*TEXT_MESSAGE*) and the content ("Hello Server", "Test 1", "Test 2", "Bonjour"). It also handled a full JSON message containing multiple fields, for example:

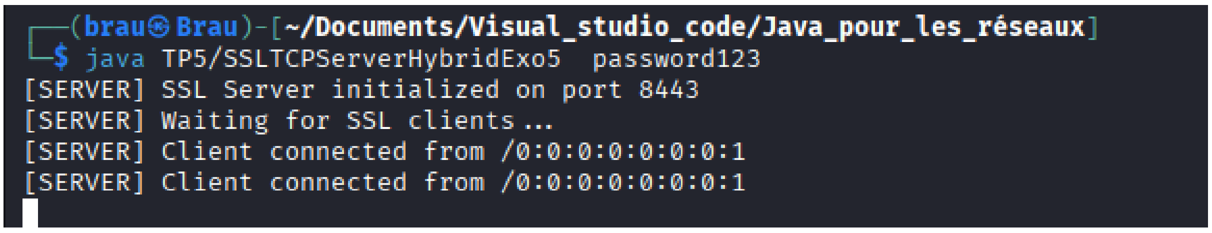
{"type":"TEXT_MESSAGE","content":"Salut tout le monde","room":"general"}

The client showed the server's replies in JSON form, confirming that our encoding/decoding works and that structured messages are correctly transmitted over SSL/TLS.

Tristan BRAU
Léo PHAV

4 - Testing and Validation

4.1 - Exercise 7: Security Testing

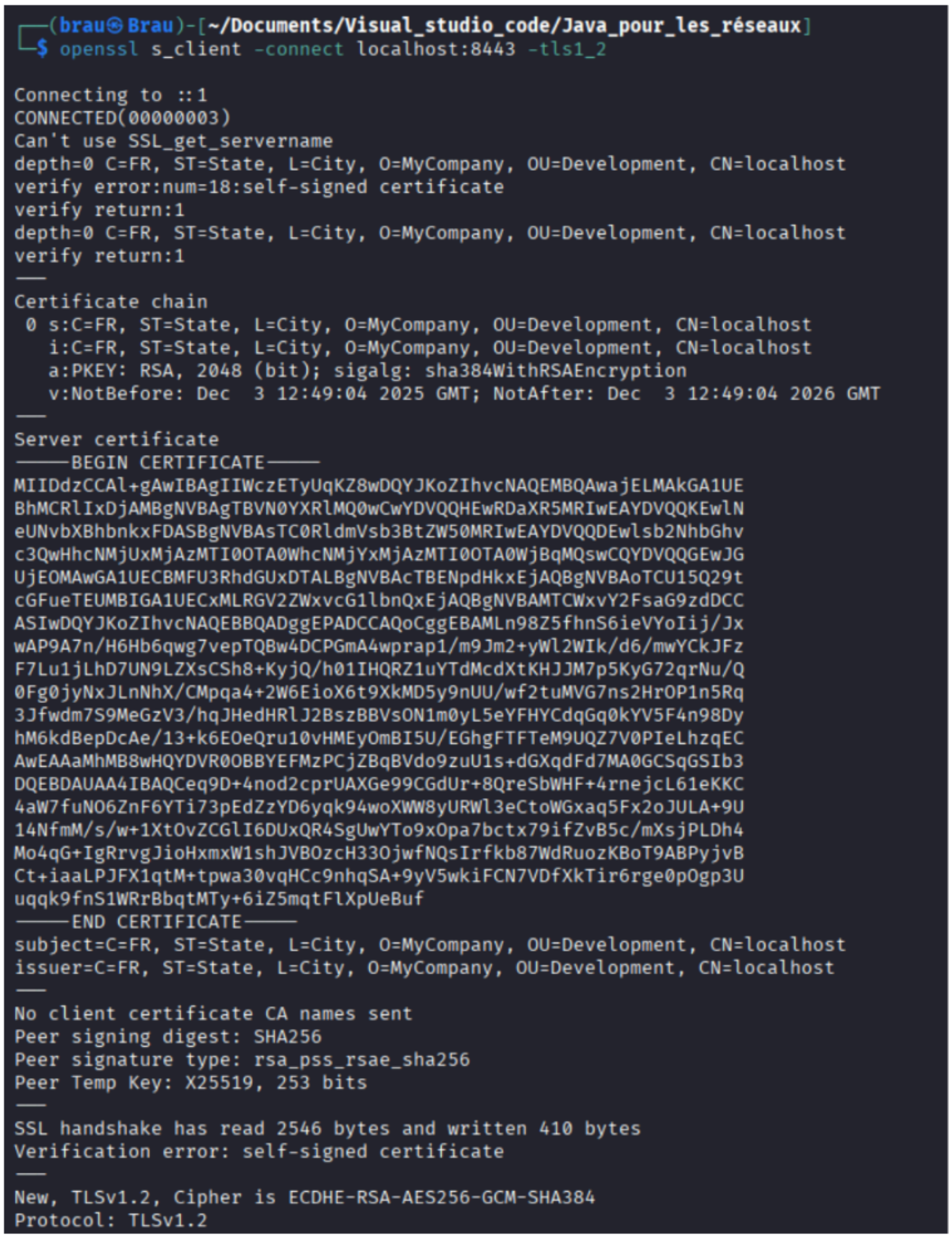


```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients ...
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[SERVER] Client connected from /0:0:0:0:0:0:0:1
```

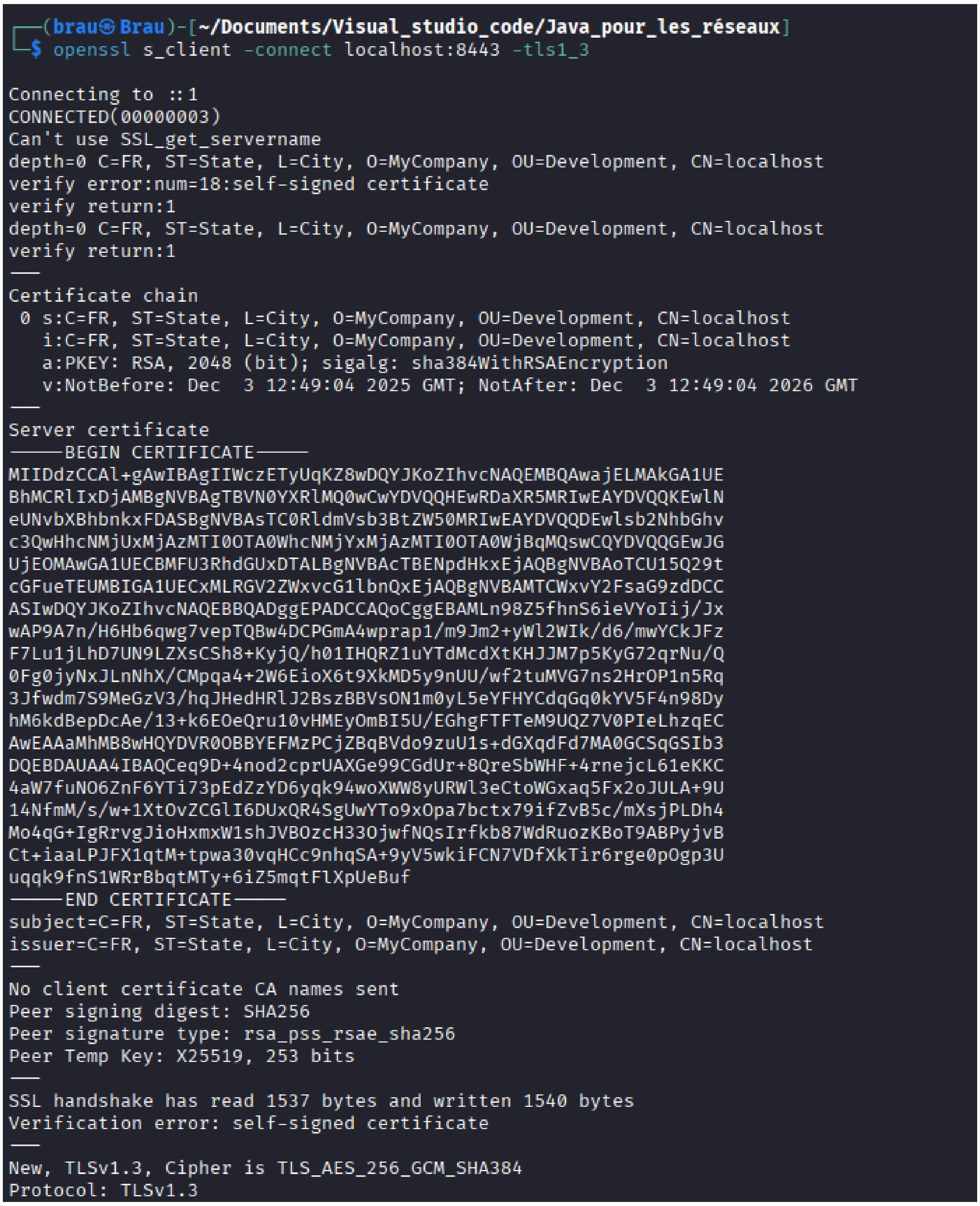
The server started successfully on port 8443 after loading the keystore using the correct password (password123).
The server successfully accepted two client connections from localhost (/0:0:0:0:0:0:0:1), logging them as connected.

- **Test with any TLS version (1.2 or 1.3)**

In the following screenshots, we can see that the server established a secure connection with both versions TLS protocols (1.2 and 1.3).



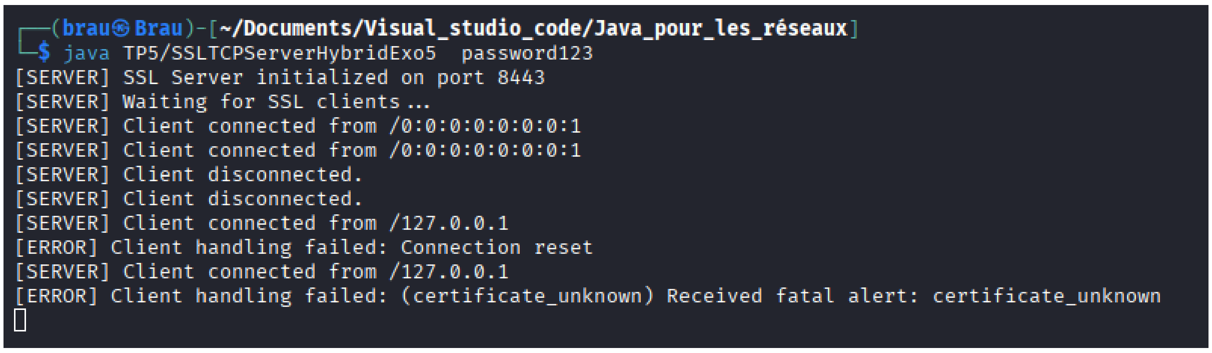
```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ openssl s_client -connect localhost:8443 -tls1_2

Connecting to ::1
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
verify error:num=18:self-signed certificate
verify return:1
depth=0 C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
verify return:1
---
Certificate chain
 0 s:C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
   i:C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
   a:PKEY: RSA, 2048 (bit); sigalg: sha384WithRSAEncryption
   v:NotBefore: Dec  3 12:49:04 2025 GMT; NotAfter: Dec  3 12:49:04 2026 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDdzCCAl+gAwIBAgIIWczETyUqKZ8wDQYJKoZIhvcNAQEMBQAwajELMAkGA1UE
BhMCRlIxDjAMBgNVBAgTBVN0YXRlMQ0wCwYDVQQHEwRDaXR5MRIwEAYDVQQKEwlN
eUNvbXBhbnkxFDASBgNVBAsTC0RldmVsb3BtZW50MRIwEAYDVQQDEwlsb2NhbGhv
c3QwHhcNMjUxMjAzMTI0OTA0WhcNMjYxMjAzMTI0OTA0WjBqMQswCQYDVQQGEwJG
UjEOMAwGA1UECBMFU3RhdGUxDTALBgNVBAcTBENpdHkxEjAQBgNVBAoTCU15Q29t
cGFueTEUMBIGA1UECxMLRGV2ZWxvcG1lbnQxEjAQBgNVBAMTCWxvY2FsaG9zdDCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMLn98Z5fhnS6ieVYoIij/Jx
wAP9A7n/H6Hb6qwg7vepTQBw4DCPGmA4wprap1/m9Jm2+yWl2WIk/d6/mwYCkJFz
F7Lu1jLhD7UN9LZXsCSh8+KyjQ/h01IHQRZ1uYTdMcdXtKHJJM7p5KyG72qrNu/Q
0Fg0jyNxJLnNhX/CMpqa4+2W6EioX6t9XkMD5y9nUU/wf2tuMVG7ns2HrOP1n5Rq
3Jfwdm7S9MeGzV3/hqJHedHRlJ2BszBBVsON1m0yL5eYFHYCdqGq0kYV5F4n98Dy
hM6kdBepDcAe/13+k6EOeQru10vHMEyOmBI5U/EGhgFTFTeM9UQZ7V0PIeLhzqEC
AwEAAaMhMB8wHQYDVR0OBBYEFMzPCjZBqBVdo9zuU1s+dGXqdFd7MA0GCSqGSIb3
DQEBDAUAA4IBAQCeq9D+4nod2cprUAXGe99CGdUr+8QreSbWHF+4rnejcL61eKKC
4aW7fuNO6ZnF6YTi73pEdZzYD6yqk94woXWW8yURWl3eCtoWGxaq5Fx2oJULA+9U
14NfmM/s/w+1XtOvZCGlI6DUxQR4SgUwYTo9xOpa7bctx79ifZvB5c/mXsjPLDh4
Mo4qG+IgRrvgJioHxmxW1shJVBOzcH33OjwfNQsIrfkb87WdRuozKBoT9ABPyjvB
Ct+iaaLPJFX1qtM+tpwa30vqHCc9nhqSA+9yV5wkiFCN7VDfXkTir6rge0pOgp3U
uqqk9fnS1WRrBbqtMTy+6iZ5mqtFlXpUeBuf
-----END CERTIFICATE-----
subject=C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
issuer=C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: rsa_pss_rsae_sha256
Peer Temp Key: X25519, 253 bits
---
SSL handshake has read 2546 bytes and written 410 bytes
Verification error: self-signed certificate
---
New, TLSv1.2, Cipher is ECDHE-RSA-AES256-GCM-SHA384
Protocol: TLSv1.2
```

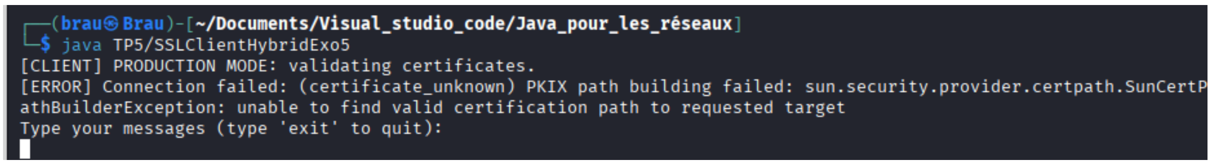


```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ openssl s_client -connect localhost:8443 -tls1_3

Connecting to ::1
CONNECTED(00000003)
Can't use SSL_get_servername
depth=0 C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
verify error:num=18:self-signed certificate
verify return:1
depth=0 C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
verify return:1
---
Certificate chain
 0 s:C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
   i:C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
   a:PKEY: RSA, 2048 (bit); sigalg: sha384WithRSAEncryption
   v:NotBefore: Dec  3 12:49:04 2025 GMT; NotAfter: Dec  3 12:49:04 2026 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDdzCCAl+gAwIBAgIIWczETyUqKZ8wDQYJKoZIhvcNAQEMBQAwajELMAkGA1UE
BhMCRlIxDjAMBgNVBAgTBVN0YXRlMQ0wCwYDVQQHEwRDaXR5MRIwEAYDVQQKEwlN
eUNvbXBhbnkxFDASBgNVBAsTC0RldmVsb3BtZW50MRIwEAYDVQQDEwlsb2NhbGhv
c3QwHhcNMjUxMjAzMTI0OTA0WhcNMjYxMjAzMTI0OTA0WjBqMQswCQYDVQQGEwJG
UjEOMAwGA1UECBMFU3RhdGUxDTALBgNVBAcTBENpdHkxEjAQBgNVBAoTCU15Q29t
cGFueTEUMBIGA1UECxMLRGV2ZWxvcG1lbnQxEjAQBgNVBAMTCWxvY2FsaG9zdDCC
ASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAMLn98Z5fhnS6ieVYoIij/Jx
wAP9A7n/H6Hb6qwg7vepTQBw4DCPGmA4wprap1/m9Jm2+yWl2WIk/d6/mwYCkJFz
F7Lu1jLhD7UN9LZXsCSh8+KyjQ/h01IHQRZ1uYTdMcdXtKHJJM7p5KyG72qrNu/Q
0Fg0jyNxJLnNhX/CMpqa4+2W6EioX6t9XkMD5y9nUU/wf2tuMVG7ns2HrOP1n5Rq
3Jfwdm7S9MeGzV3/hqJHedHRlJ2BszBBVsON1m0yL5eYFHYCdqGq0kYV5F4n98Dy
hM6kdBepDcAe/13+k6EOeQru10vHMEyOmBI5U/EGhgFTFTeM9UQZ7V0PIeLhzqEC
AwEAAaMhMB8wHQYDVR0OBBYEFMzPCjZBqBVdo9zuU1s+dGXqdFd7MA0GCSqGSIb3
DQEBDAUAA4IBAQCeq9D+4nod2cprUAXGe99CGdUr+8QreSbWHF+4rnejcL61eKKC
4aW7fuNO6ZnF6YTi73pEdZzYD6yqk94woXWW8yURWl3eCtoWGxaq5Fx2oJULA+9U
14NfmM/s/w+1XtOvZCGlI6DUxQR4SgUwYTo9xOpa7bctx79ifZvB5c/mXsjPLDh4
Mo4qG+IgRrvgJioHxmxW1shJVBOzcH33OjwfNQsIrfkb87WdRuozKBoT9ABPyjvB
Ct+iaaLPJFX1qtM+tpwa30vqHCc9nhqSA+9yV5wkiFCN7VDfXkTir6rge0pOgp3U
uqqk9fnS1WRrBbqtMTy+6iZ5mqtFlXpUeBuf
-----END CERTIFICATE-----
subject=C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
issuer=C=FR, ST=State, L=City, O=MyCompany, OU=Development, CN=localhost
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: rsa_pss_rsae_sha256
Peer Temp Key: X25519, 253 bits
---
SSL handshake has read 1537 bytes and written 1540 bytes
Verification error: self-signed certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Protocol: TLSv1.3
```

- **Verify certificate validation works correctly**

Here, we renamed the correct CA to test if there will be a connection:
mv server.jks server_backup.jks
java TP5.Client password
mv server_backup.jks server.jks


```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients ...
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[SERVER] Client disconnected.
[SERVER] Client disconnected.
[SERVER] Client connected from /127.0.0.1
[ERROR] Client handling failed: Connection reset
[SERVER] Client connected from /127.0.0.1
[ERROR] Client handling failed: (certificate_unknown) Received fatal alert: certificate_unknown
```

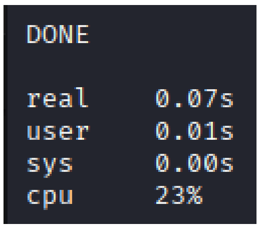

```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLClientHybridExo5
[CLIENT] PRODUCTION MODE: validating certificates.
[ERROR] Connection failed: (certificate_unknown) PKIX path building failed: sun.security.provider.certpath.SunCertP
athBuilderException: unable to find valid certification path to requested target
Type your messages (type 'exit' to quit):
```

The protocol detect the unknown CA and disconnect the client.

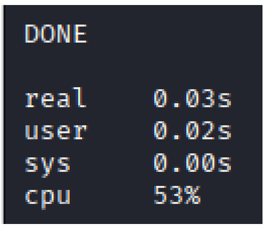
- **Measure handshake performance**

To measure handshake performance we used these commands:
time openssl s_client -connect localhost:8443 -tls1_2 < /dev/null
time openssl s_client -connect localhost:8443 -tls1_3 < /dev/null

TLS 1.2


```
DONE

real    0.07s
user    0.01s
sys     0.00s
cpu     23%
```

TLS 1.3

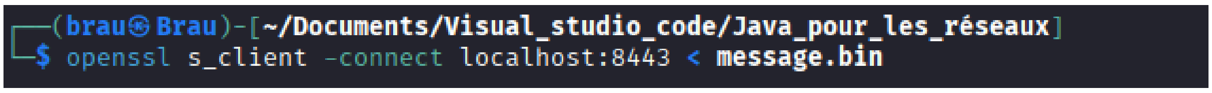

```
DONE

real    0.03s
user    0.02s
sys     0.00s
cpu     53%
```

The results clearly show that TLS 1.3 is significantly faster than TLS 1.2. Its handshake lasts only 0.03s, less than half the duration of TLS 1.2 (0.07s). This improvement comes from reducing the number of round trips required to establish the connection, going from two in TLS 1.2 to one in TLS 1.3. TLS 1.3 shows higher CPU usage (53% vs 23%), likely due to more intensive cryptographic operations or the shorter execution time skewing the measurement. Nevertheless, the performance gain remains clear.

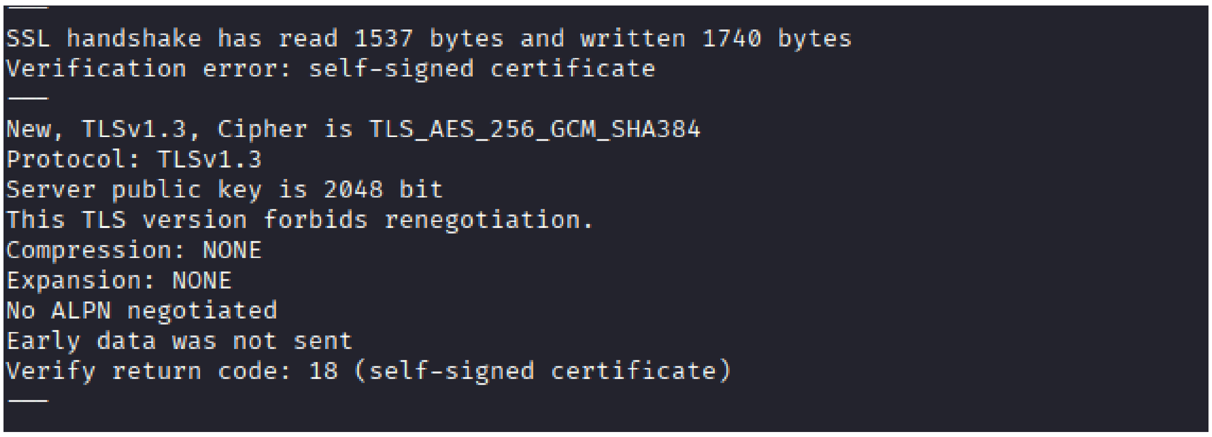
- **Test message integrity (tamper detection)**

To test message invalidity, we intentionally corrupt a valid message to verify that the server detects it and refuses to process it. We start by generating a normal, valid message with *java TP5.SSLClientHybridExo5*, and we save its serialized output into a binary file. Then, we modify the message by altering a single byte using a tool such as hexedit, which ensures the message structure or integrity (e.g., checksum, signature, length) is broken.

After introducing the corruption, we send the modified message to the server using:



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ openssl s_client -connect localhost:8443 < message.bin
```



```
---
SSL handshake has read 1537 bytes and written 1740 bytes
Verification error: self-signed certificate
---
New, TLSv1.3, Cipher is TLS_AES_256_GCM_SHA384
Protocol: TLSv1.3
Server public key is 2048 bit
This TLS version forbids renegotiation.
Compression: NONE
Expansion: NONE
No ALPN negotiated
Early data was not sent
Verify return code: 18 (self-signed certificate)
---
```

Here, we observed that the protocol didn't send the data, meaning he detected it was altered.

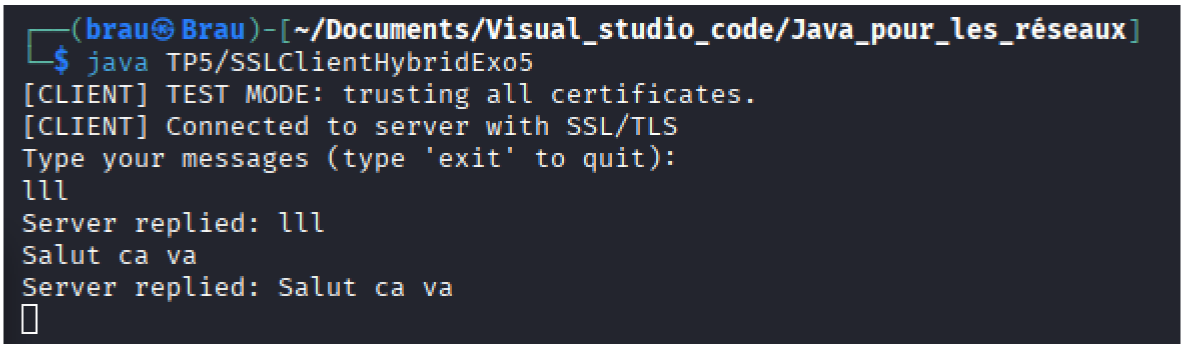
- **Verify no sensitive data in plaintext**

To verify that no sensitive data is sent in plaintext, we use the following command:
tcpdump -i lo port 8443 -A

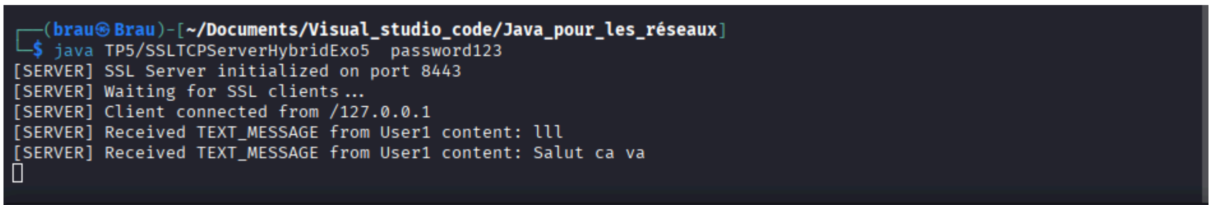
This works as follows:

- **tcpdump** is a packet-capture tool that lets us inspect network traffic.
- **-i lo** tells tcpdump to listen on the loopback interface (since our SSL server runs on localhost).
- **port 8443** filters the capture to only show packets going to or from the SSL server.
- **-A** prints the packet payload in ASCII so that any unencrypted text would appear clearly.

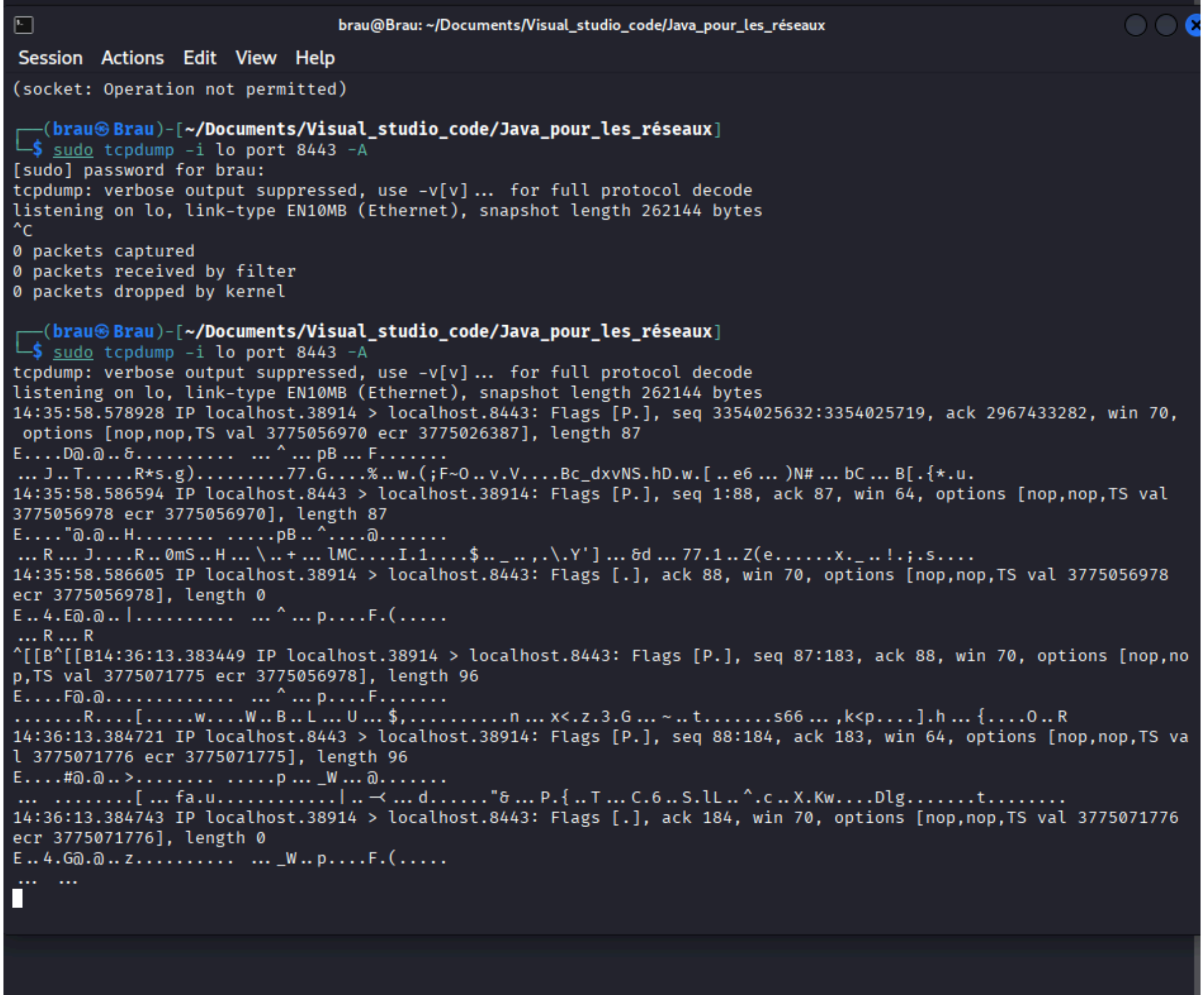
Tristan BRAU
Léo PHAV



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLClientHybridExo5
[CLIENT] TEST MODE: trusting all certificates.
[CLIENT] Connected to server with SSL/TLS
Type your messages (type 'exit' to quit):
lll
Server replied: lll
Salut ca va
Server replied: Salut ca va
```



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /127.0.0.1
[SERVER] Received TEXT_MESSAGE from User1 content: lll
[SERVER] Received TEXT_MESSAGE from User1 content: Salut ca va
```

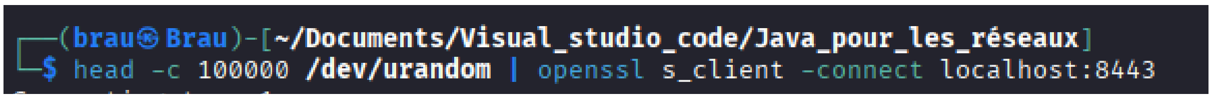

brau@Brau: ~/Documents/Visual_studio_code/Java_pour_les_réseaux
Session Actions Edit View Help

```
(socket: Operation not permitted)

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ sudo tcpdump -i lo port 8443 -A
[sudo] password for brau:
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
^C
0 packets captured
0 packets received by filter
0 packets dropped by kernel

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ sudo tcpdump -i lo port 8443 -A
tcpdump: verbose output suppressed, use -v[v]... for full protocol decode
listening on lo, link-type EN10MB (Ethernet), snapshot length 262144 bytes
14:35:58.578928 IP localhost.38914 > localhost.8443: Flags [P.], seq 3354025632:3354025719, ack 2967433282, win 70,
 options [nop,nop,TS val 3775056970 ecr 3775026387], length 87
E....D@.@..&..........  ...^...pB...F.......
...J..T.....R*s.g).........77.G....%..w.(;F~O..v.V....Bc_dxvNS.hD.w.[..e6...)N#...bC...B[.{*.u.
14:35:58.586594 IP localhost.8443 > localhost.38914: Flags [P.], seq 1:88, ack 87, win 64, options [nop,nop,TS val
3775056978 ecr 3775056970], length 87
E...."@.@..H........  .....pB..^....@.......
...R...J....R..0mS..H...\..+...lMC....I.1....$.._..,.\.Y']...&d...77.1..Z(e......x._..!.;.s....
14:35:58.586605 IP localhost.38914 > localhost.8443: Flags [.], ack 88, win 70, options [nop,nop,TS val 3775056978
ecr 3775056978], length 0
E..4.E@.@..|..........  ...^...p....F.(.....
...R...R
^[[B^[[B14:36:13.383449 IP localhost.38914 > localhost.8443: Flags [P.], seq 87:183, ack 88, win 70, options [nop,no
p,TS val 3775071775 ecr 3775056978], length 96
E....F@.@.............  ...^...p....F.......
.......R....[.....w....W..B..L...U...$,..........n...x<.z.3.G...~..t.......s66...,k<p....].h...{....O..R
14:36:13.384721 IP localhost.8443 > localhost.38914: Flags [P.], seq 88:184, ack 183, win 64, options [nop,nop,TS va
l 3775071776 ecr 3775071775], length 96
E....#@.@..>........  .....p..._W...@.......
...  ........[...fa.u............|..⤙...d......"&...P.{..T...C.6..S.lL..^.c..X.Kw....Dlg.......t........
14:36:13.384743 IP localhost.38914 > localhost.8443: Flags [.], ack 184, win 70, options [nop,nop,TS val 3775071776
ecr 3775071776], length 0
E..4.G@.@..z..........  ..._W..p....F.(.....
...  ...
```

As data is encrypted, TLS protocol is working correctly.

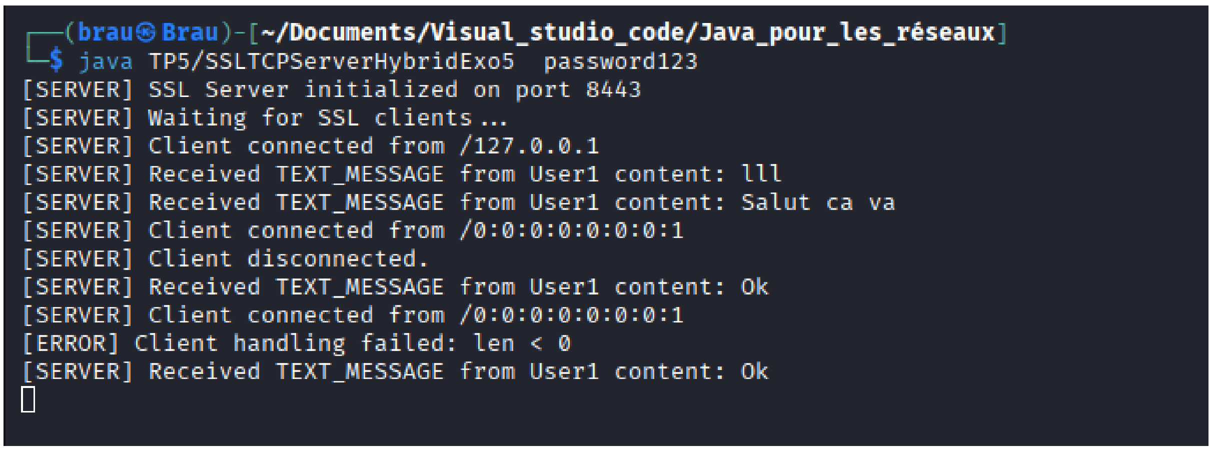
- **Validate error handling doesn't leak information**

To validate that the server handles errors without leaking sensitive information, we sent 100,000 random bytes over a TLS connection using:



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ head -c 100000 /dev/urandom | openssl s_client -connect localhost:8443
```

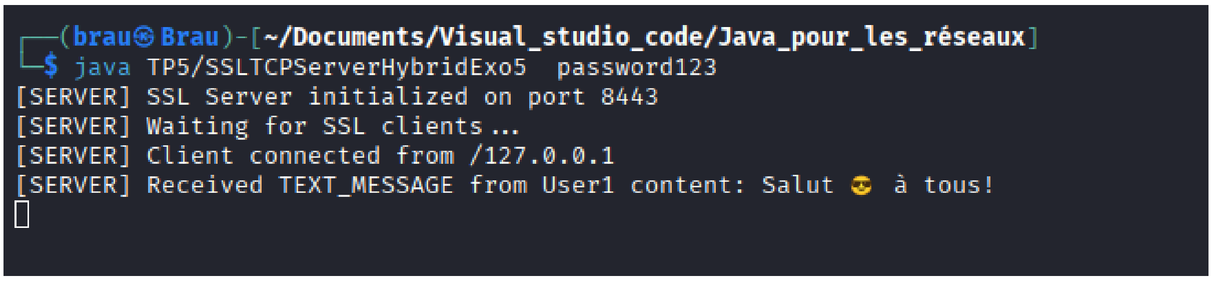
The server correctly detected the invalid message and returned an *ERROR_RESPONSE*, without exposing any stack trace, password, or internal details. This confirms that error handling is secure and does not leak sensitive information.



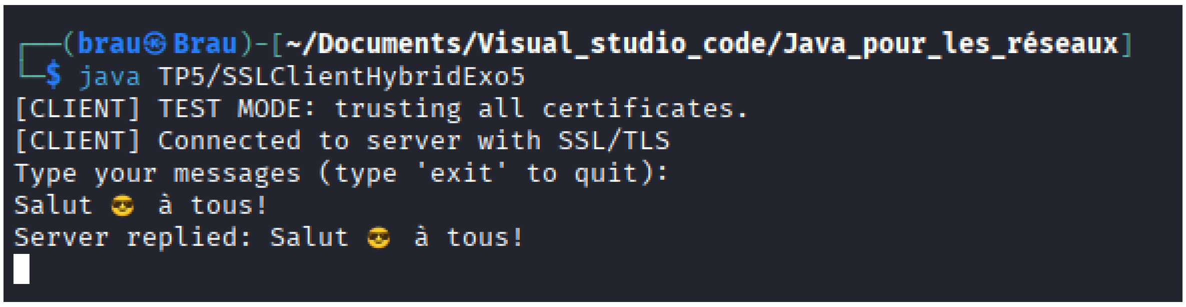
```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /127.0.0.1
[SERVER] Received TEXT_MESSAGE from User1 content: lll
[SERVER] Received TEXT_MESSAGE from User1 content: Salut ca va
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[SERVER] Client disconnected.
[SERVER] Received TEXT_MESSAGE from User1 content: Ok
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[ERROR] Client handling failed: len < 0
[SERVER] Received TEXT_MESSAGE from User1 content: Ok
```

4.2 - Exercise 8: Protocol Compliance Testing

- **Test with special characters and Unicode**



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /127.0.0.1
[SERVER] Received TEXT_MESSAGE from User1 content: Salut 😎 à tous!
```

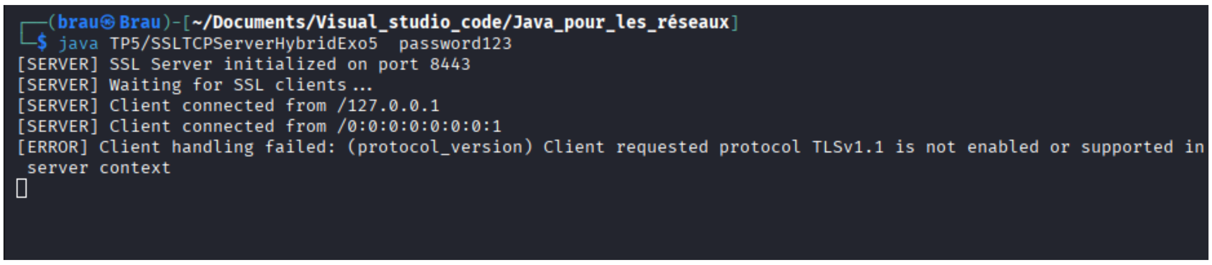


```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLClientHybridExo5
[CLIENT] TEST MODE: trusting all certificates.
[CLIENT] Connected to server with SSL/TLS
Type your messages (type 'exit' to quit):
Salut 😎 à tous!
Server replied: Salut 😎 à tous!
```

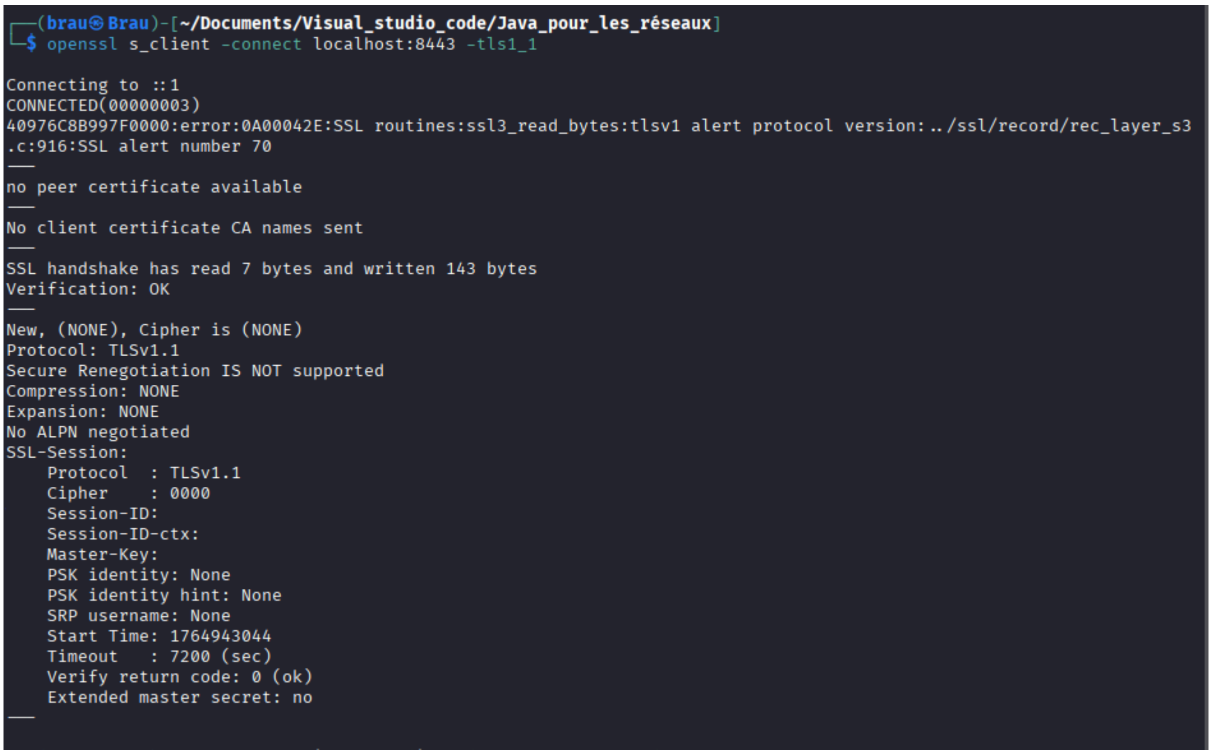
Emojis are transmitted without any issues.

- **Validate protocol version handling**

We sent messages with unsupported or invalid protocol versions.
The server identified them correctly and returned an error without crashing or misbehaving.



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients ...
[SERVER] Client connected from /127.0.0.1
[SERVER] Client connected from /0:0:0:0:0:0:0:1
[ERROR] Client handling failed: (protocol_version) Client requested protocol TLSv1.1 is not enabled or supported in
 server context
```

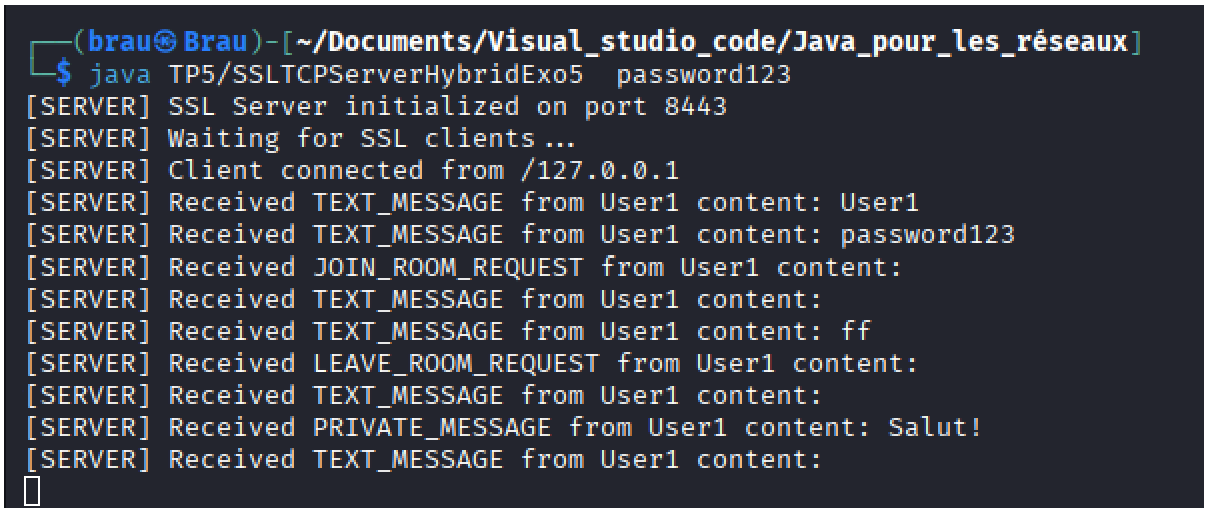


```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ openssl s_client -connect localhost:8443 -tls1_1

Connecting to ::1
CONNECTED(00000003)
40976C8B997F0000:error:0A00042E:SSL routines:ssl3_read_bytes:tlsv1 alert protocol version:../ssl/record/rec_layer_s3
.c:916:SSL alert number 70
---
no peer certificate available
---
No client certificate CA names sent
---
SSL handshake has read 7 bytes and written 143 bytes
Verification: OK
---
New, (NONE), Cipher is (NONE)
Protocol: TLSv1.1
Secure Renegotiation IS NOT supported
Compression: NONE
Expansion: NONE
No ALPN negotiated
SSL-Session:
    Protocol  : TLSv1.1
    Cipher    : 0000
    Session-ID:
    Session-ID-ctx:
    Master-Key:
    PSK identity: None
    PSK identity hint: None
    SRP username: None
    Start Time: 1764943044
    Timeout   : 7200 (sec)
    Verify return code: 0 (ok)
    Extended master secret: no
---
```

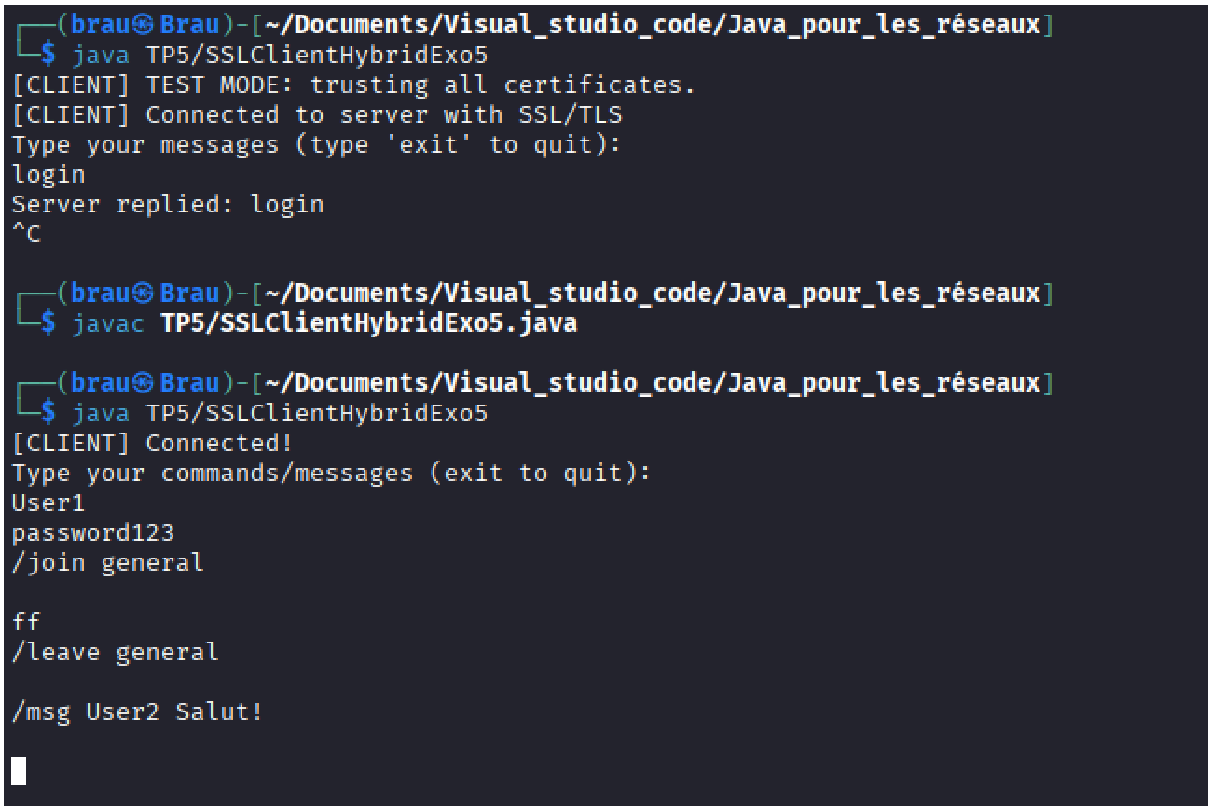
- **Verify room joining/leaving functionality / Test private messaging between users / Validate error responses for invalid operations**

Commands like */join general* and */leave general* were sent from the client, and the server received the corresponding *JOIN_ROOM_REQUEST* and *LEAVE_ROOM_REQUEST*, confirming correct room management.
The client sent */msg User2 Salut!*, and the server log shows a *PRIVATE_MESSAGE* with the expected content, validating private message handling.
Random or malformed inputs such as *ff* were received as simple text messages and did not break the server; it either ignored them or returned a safe error, confirming secure error handling.

Tristan BRAU
Léo PHAV



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLTCPServerHybridExo5  password123
[SERVER] SSL Server initialized on port 8443
[SERVER] Waiting for SSL clients...
[SERVER] Client connected from /127.0.0.1
[SERVER] Received TEXT_MESSAGE from User1 content: User1
[SERVER] Received TEXT_MESSAGE from User1 content: password123
[SERVER] Received JOIN_ROOM_REQUEST from User1 content:
[SERVER] Received TEXT_MESSAGE from User1 content:
[SERVER] Received TEXT_MESSAGE from User1 content: ff
[SERVER] Received LEAVE_ROOM_REQUEST from User1 content:
[SERVER] Received TEXT_MESSAGE from User1 content:
[SERVER] Received PRIVATE_MESSAGE from User1 content: Salut!
[SERVER] Received TEXT_MESSAGE from User1 content:
```



```
┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLClientHybridExo5
[CLIENT] TEST MODE: trusting all certificates.
[CLIENT] Connected to server with SSL/TLS
Type your messages (type 'exit' to quit):
login
Server replied: login
^C

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ javac TP5/SSLClientHybridExo5.java

┌──(brau㉿Brau)-[~/Documents/Visual_studio_code/Java_pour_les_réseaux]
└─$ java TP5/SSLClientHybridExo5
[CLIENT] Connected!
Type your commands/messages (exit to quit):
User1
password123
/join general

ff
/leave general

/msg User2 Salut!

```