Report (Shynbolat Unaibaev)

1. Objectives

In this work we aimed to gain practical experience with the TLS protocol. The tasks included:

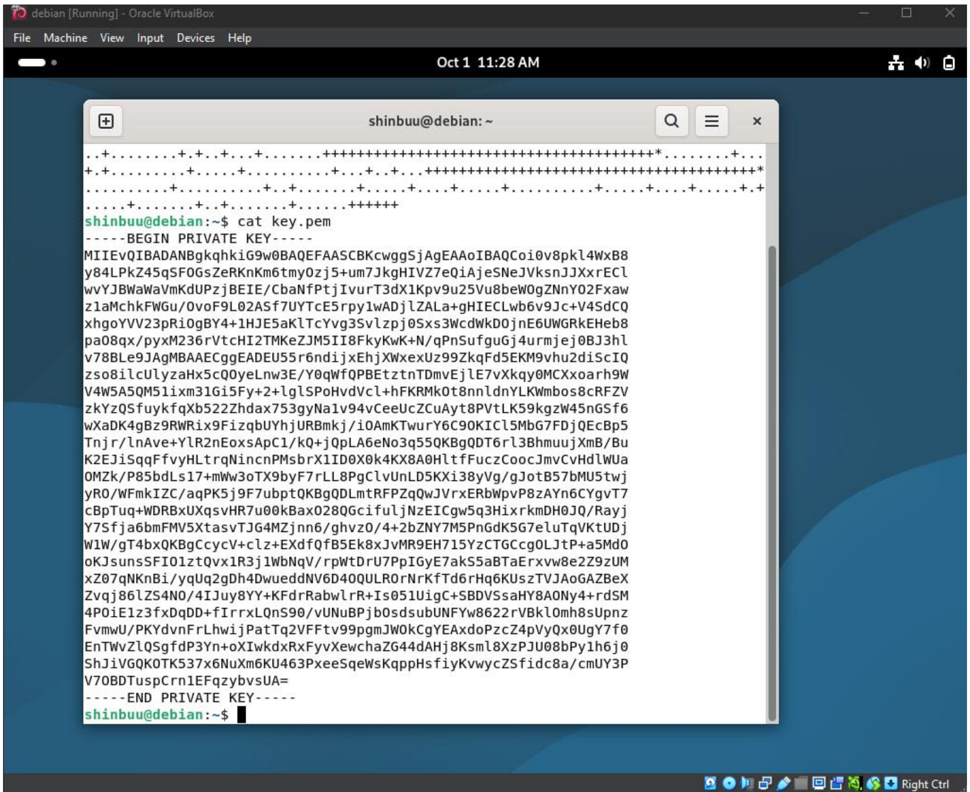
- generating a key and certificate using OpenSSL,
- running a TLS server and client,
- analyzing the handshake process in Wireshark,
- experimenting with cipher suite restrictions, TLS versions, and invalid certificates,
- making conclusions about secure communication.

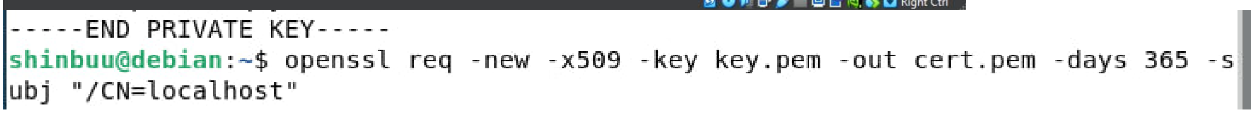
2. Preparation

- We used **Debian Linux**, where OpenSSL is already built-in.
- Installed **Wireshark** for packet analysis.
- OpenSSL already in Debian

3. Key and Certificate Generation

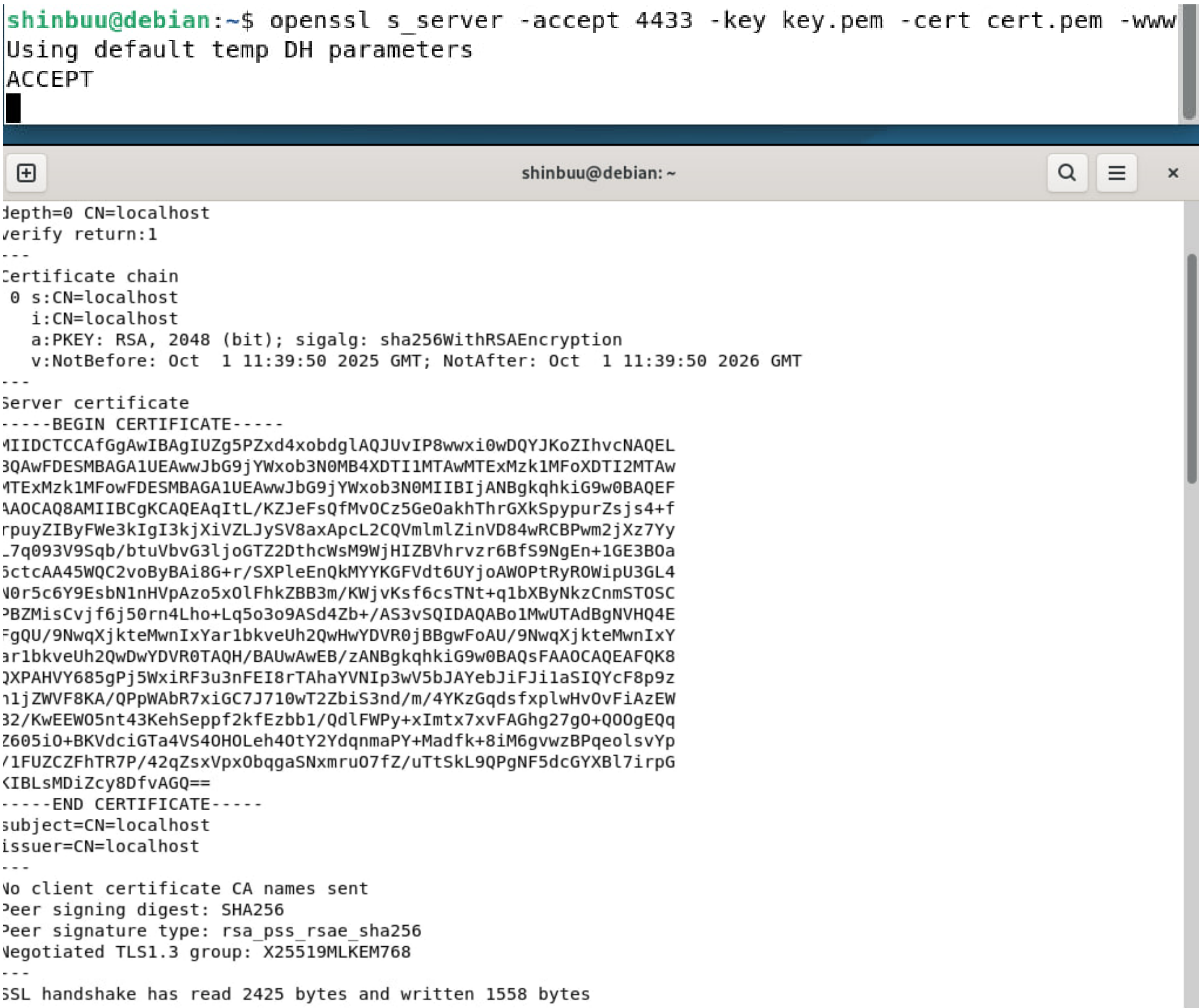
- First, a private RSA key (2048 bits) was generated.
- Then, a self-signed X.509 certificate valid for 365 days was created.
  This certificate was required for the server, so the client could verify its authenticity.





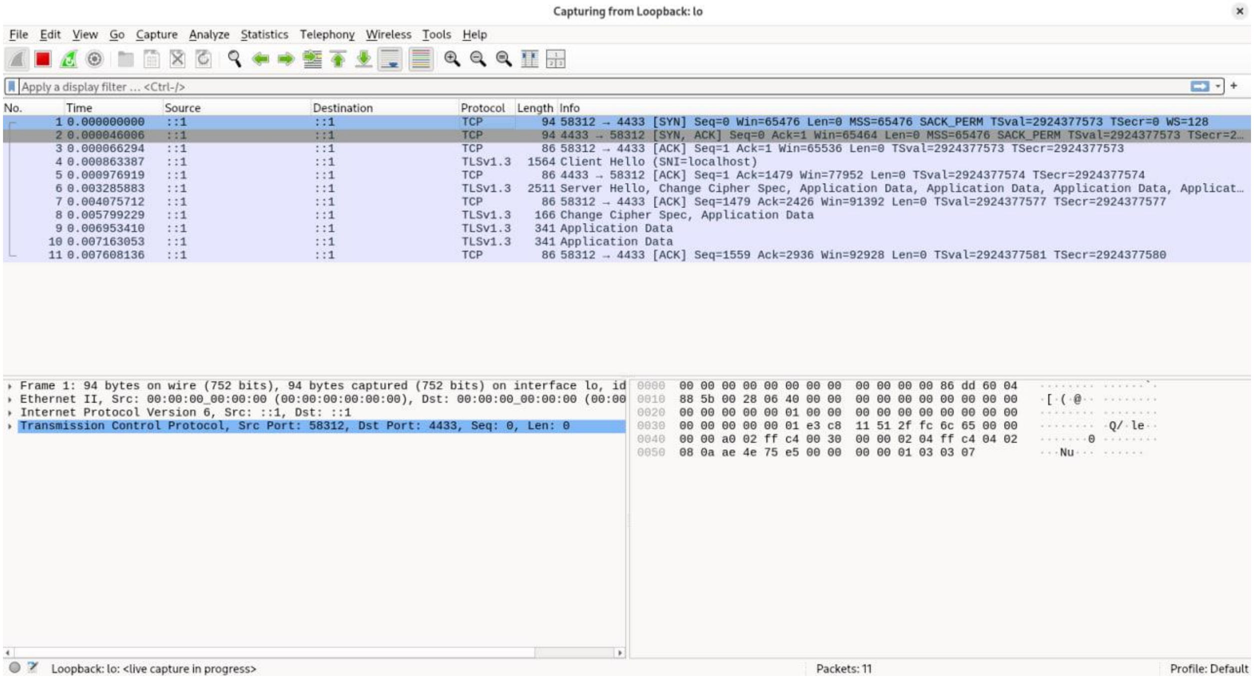
4. Server and Client Setup

- The TLS server was launched on port 4433 using the generated key and certificate.
- The client (`openssl s_client`) connected to the server.
- The client output displayed the certificate, the chosen cipher suite, and the TLS version (TLS 1.3).



5. Handshake Analysis in Wireshark

- In Wireshark, the **lo (loopback)** interface was selected because the connection was made via `127.0.0.1`.
- Packets were filtered by port 4433.
- The following sequence was observed:
    1. **ClientHello** – client offered a list of supported cipher suites.
    2. **ServerHello** – server selected one cipher suite.
    3. **Certificate** – server sent its certificate.
    4. **Finished** – handshake completion.



In TLS 1_3 Certificate and Finished are ecnrypted compared to TLS 1_2

```
---
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)
---
---
Post-Handshake New Session Ticket arrived:
SSL-Session:
    Protocol  : TLSv1.3
    Cipher    : TLS_AES_256_GCM_SHA384
    Session-ID: EFBD635165F1CFCE7C56324CBDC196D35E7B8C6E986D4C26046791B9075F8B8D
    Session-ID-ctx:
    Resumption PSK: 3197975AEAF91403D95E704EB5174E74096ADD24028666D63C3F9C45858F
27F23B6202D13F590D22786341357C220F97
    PSK identity: None
    PSK identity hint: None
```

6. Experiments

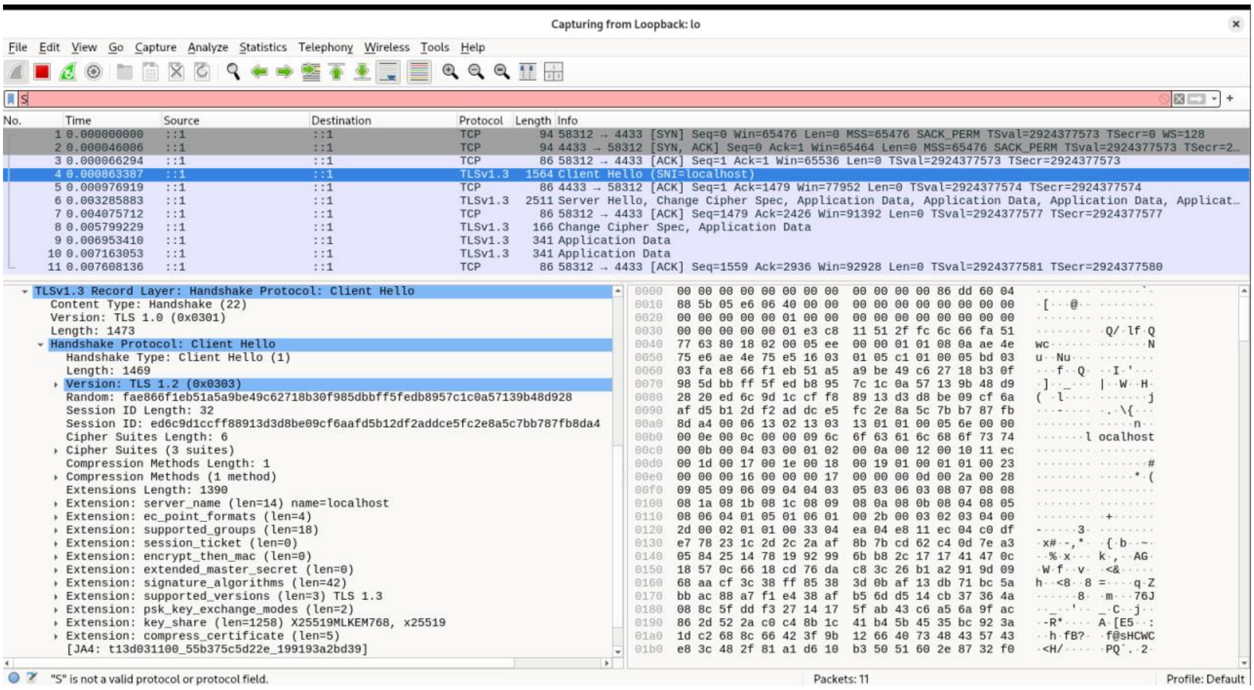
6.1 Restricting Cipher Suites

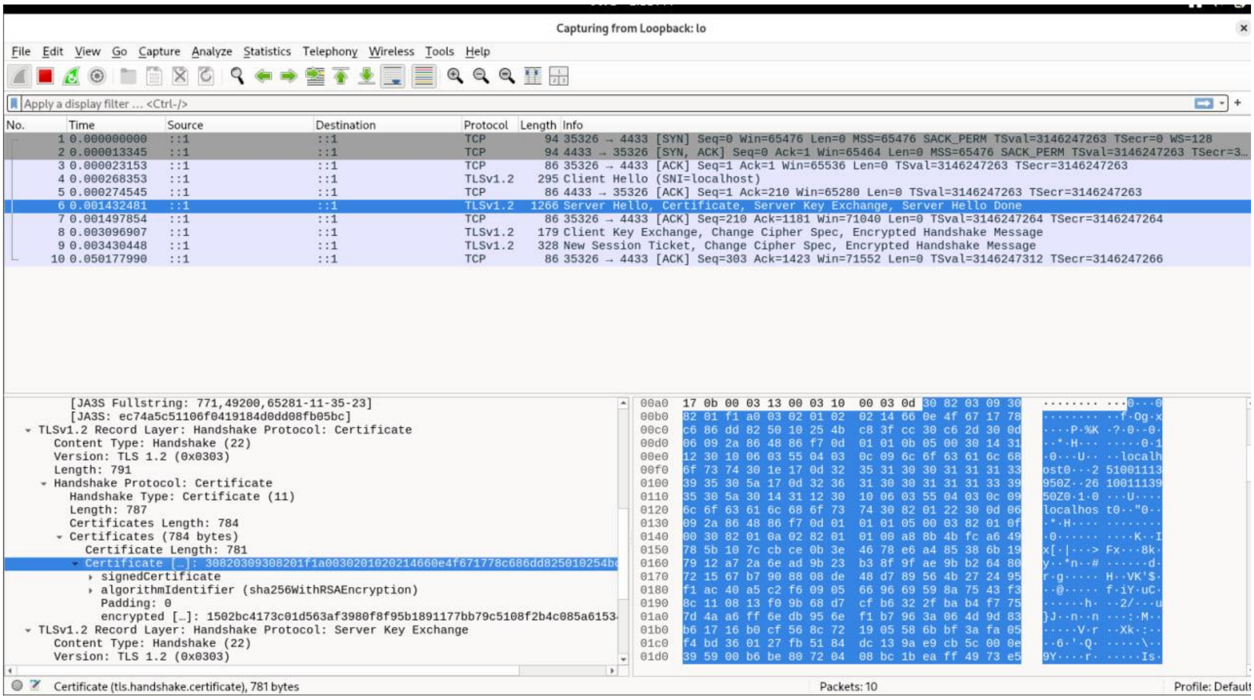
The server was started with a single cipher suite (`ECDHE-RSA-AES256-GCM-SHA384`). As a result, the server always chose this suite. This demonstrated how the server controls algorithm selection.

```
shinbuu@debian:~$ openssl s_server -accept 4433 -key key.pem -cert cert.pem -cipher 'ECDHE
-RSA-AES256-GCM-SHA384'
Using default temp DH parameters
ACCEPT
-----BEGIN SSL SESSION PARAMETERS-----
MIGEAgEBAgIDBAQCEwIEIKNGPzNcHyABUf3gnMLfZwHB6m0wwlKF20gmrMIU1kjw
BDD8Z2Ef6YYnwICg2f3cDkJnYeqSC7HrFoUZOAw8RQA2ybeQngzjlZIFpUzr8l9g
9O+hBgIEaN03UqIEAgIcIKQGBAQBAAAArgcCBQC1RBPiswQCAhHs
-----END SSL SESSION PARAMETERS-----
Shared ciphers:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256
Signature Algorithms: id-ml-dsa-65:id-ml-dsa-87:id-ml-dsa-44:ECDSA+SHA256:ECDSA+SHA384:ECD
SA+SHA512:ed25519:ed448:ecdsa_brainpoolP256r1_sha256:ecdsa_brainpoolP384r1_sha384:ecdsa_br
ainpoolP512r1_sha512:rsa_pss_pss_sha256:rsa_pss_pss_sha384:rsa_pss_pss_sha512:RSA-PSS+SHA2
56:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512
Shared Signature Algorithms: id-ml-dsa-65:id-ml-dsa-87:id-ml-dsa-44:ECDSA+SHA256:ECDSA+SHA
384:ECDSA+SHA512:ed25519:ed448:ecdsa_brainpoolP256r1_sha256:ecdsa_brainpoolP384r1_sha384:e
cdsa_brainpoolP512r1_sha512:rsa_pss_pss_sha256:rsa_pss_pss_sha384:rsa_pss_pss_sha512:RSA-P
SS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512
Supported groups: X25519MLKEM768:x25519:secp256r1:x448:secp384r1:secp521r1:ffdhe2048:ffdhe
3072
Shared groups: X25519MLKEM768:x25519:secp256r1:x448:secp384r1:secp521r1:ffdhe2048:ffdhe307
2
CIPHER is TLS_AES_256_GCM_SHA384
```

6.2 Comparing TLS 1.2 and TLS 1.3

- With TLS 1.2, the client and server exchanged more handshake messages.
- TLS 1.3 was faster, requiring fewer steps to establish a connection and using more modern algorithms.

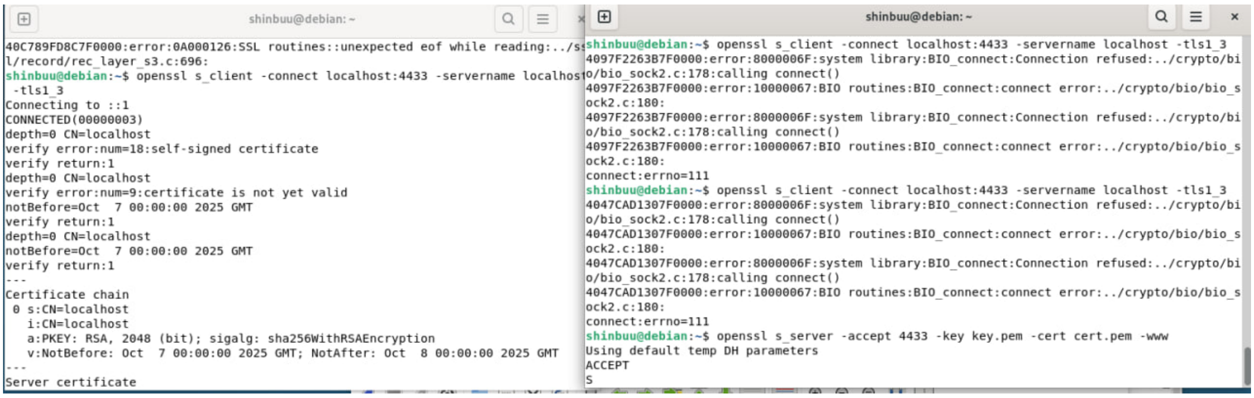




As we can see there Certificate and algorithm that was used in TLS 1_2

6.3 Invalid and Expired Certificates

A certificate with past validity dates was generated (using `faketime`). The client returned the error:



These experiments confirmed that certificate validation works as a protection against fake or outdated credentials.

7. Results and Conclusions

- Key and certificate generation was successful.
- TLS server and client established a secure connection.
- Wireshark confirmed the handshake process and algorithm selection.
- TLS 1.3 showed better performance and security compared to TLS 1.2.
- Invalid certificates resulted in verification errors, highlighting the importance of proper configuration.

8. Final Conclusion

We gained practical knowledge of TLS, including how certificates, algorithms, and protocol versions operate.

Main takeaways:

- Always use up-to-date TLS versions (TLS 1.3).
- Certificates must be valid (with correct CN and expiration dates).
- Weak or outdated cipher suites should not be used.
- Tools like OpenSSL and Wireshark are valuable for understanding the inner workings of cryptographic protocols.