

# Report (Shynbolat Unaibaev)

## 1. Objectives

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

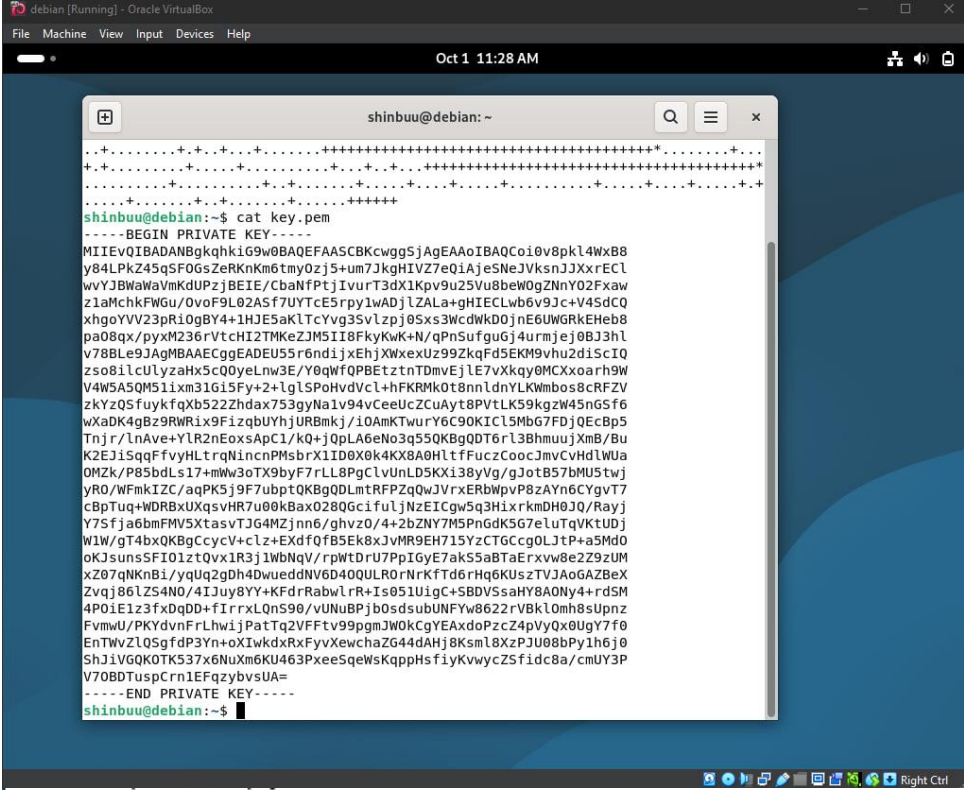
- generating a key and certificate using OpenSSL,
- running a TLS server and client,
- analyzing the handshake process in Wireshark,
- experimenting with cipher suite restrictions, TLS versions, and invalid certificates,
- making conclusions about secure communication.

## 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.



```
shinbuu@debian:~$ cat key.pem
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQFAFAASCBKcwggSjAgEAAoIBAQCoi0v8pk14WxB8
y84LPKZ45qSF0G6sZrKnm6tmy0zj5+um7JkgHIVZ7eQIAjeSNeJVksnJJXxrEC1
wYJBWwWVmKdUPzjBEIE/CbaNfPtjIvurT3dX1Kpv9u25Vu8beW0GZNNY02Fxaw
z1aMchFWGv/OvoF9L02ASf7UYTcE5rpy1wADj1ZALa+gHIECLwb6v9Jc+V45dCQ
xhgoYVV23pr10gBY4+1HJE5aKLTcYvg3SVLzpj0Sxs3WcdWkD0jnE6UWGRKEHeb8
pa08qx/pyxM236rVtchI2TMKeZJM5II8FkyKwK+n/QPnSufguGj4urmjeje0BJ3hl
v78BLE9JAgMBAAECggEAEU55r6ndijxEhjXWxexUz99ZkqFd5EKm9vhu2diScIQ
zso81lcULyzaHx5cQ0yeLnw3E/Y0qWfQPBETztnTdmvEjLE7vXkqy0MCXxoarh9W
V4W5A5QM511xm31G15fy+2+LgLS0HvdVcl+hFKRMk0t8nnldnYlKWmbos8cRFZV
zkYzQ5fuykfQxb522Zhdax753gyNa1v94vCeeUCZCuaYt8PVTLK59kgzW45nG5f6
wXaDK4gBz9RWRix9FizqbUYhJURBmkj/i0AmKTWurY6C90KICL5MbG7FDjQEcBp5
Tnjr/LnAve+YLR2nEoxsApC1/kQ+qJpLA6eNo3q55QKBgQDT6r13BhmuujXmB/Bu
K2EJiSqqFfvyHLTrqNincnPMsbrX1ID0X0k4KX8A0HltfFucZCoocJmVcVHdLUa
OMZk/P85bdLs17+mWw3oTX9byF7rLL8PgClvUnLD5KXi38yVg/gJotB57bMU5twj
yRO/WFmkIZC/eqPK5j9F7ubptQKBgQDLmtRFPZqQwJvrxERBwPvP8zAyn6CYgvT7
cBpTuq+WDRBxUXqsvHR7u00kBax028QGcifuljNzEICgw5q3HixrkmDH0JQ/Rayj
Y75fja6bmFMV5XtasvTJ64Mzjnn6/ghvz0/4+2bZNY7M5PnGdK5G7eLUtqVKtUDj
W1W/gT4bxQKBgCycv+clz+EXdfqfB5Ek8xJvMR9EH715YzCTGCG0LJtP+a5Md0
okJ3sunsFIO1ztQvx1R3j1WbNqV/rpWtdrU7PpIGyE7ak55aBtaErxxv8e2Z9ZUM
X207qNKNBi/yqUq2gDh4duweddNV6D400ULR0rNrkfTd6rHq6KUszTVJAoGAZBeX
Zvqj861ZS4N0/4IJuy8YY+KfdrRabwlrR+Is051UigC+SBDV5saHY8A0Ny4+rdSM
4P0iE1z3fxDqDD+fIrrxLQnS90/vUNuBPjb0sdsuUNFYw8622rVBkL0mh8sUpnz
FvmwU/PKYdvnFrLhw1jPatTq2VFFtv99pgmJW0KcgYEAXdoPzcZ4pVyQx0UgY7f0
EnTWvZLQ5gfdP3Yn+oXiWkdxRxFvXewchazG44dAHj8KsmL8XzPJU088bPy1h6j0
ShJiVGQ0TK537x6NuXm6KU463PxeeSqeWsKappHsf1yKvwycZ5fidc8a/cmUY3P
V70BDTuspCn1EFqzybvsUA=
-----END PRIVATE KEY-----
shinbuu@debian:~$

-----END PRIVATE KEY-----
shinbuu@debian:~$ openssl req -new -x509 -key key.pem -out cert.pem -days 365 -s
ubj "/CN=localhost"
```

## 4. Server and Client Setup

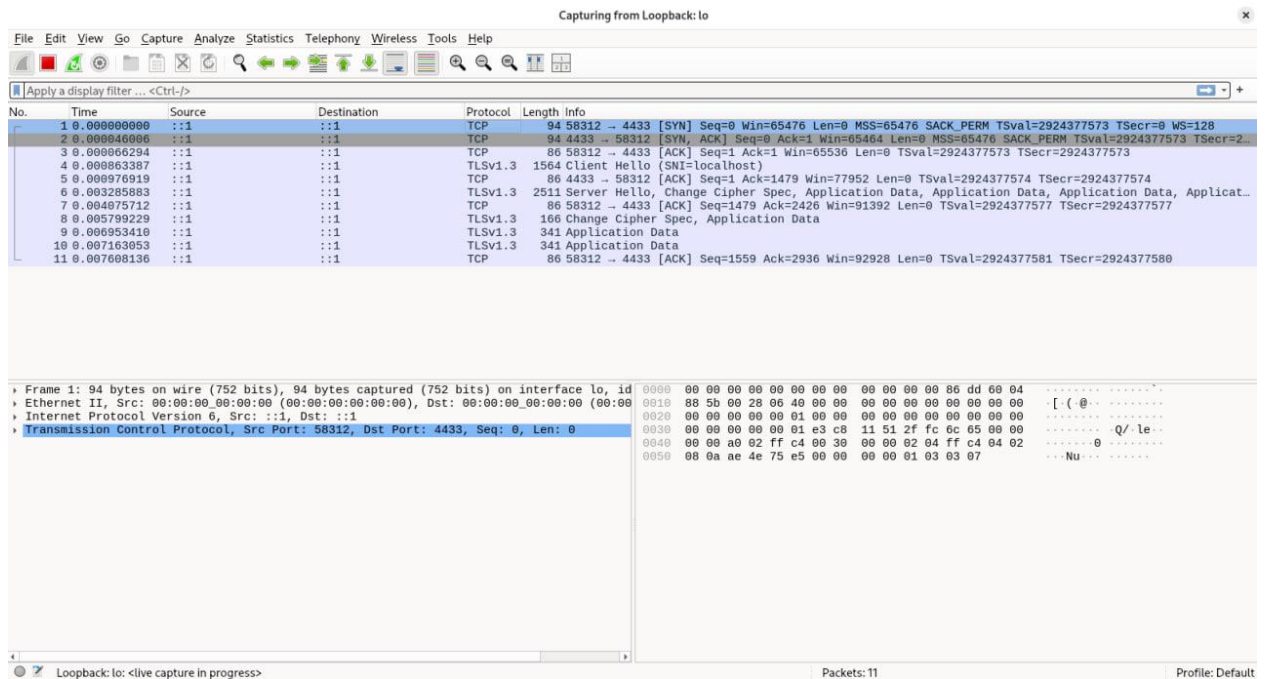
- The TLS server was launched on port 4433 using the generated key and certificate.
- The client (`openssl s_client`) connected to the server.
- The client output displayed the certificate, the chosen cipher suite, and the TLS version (TLS 1.3).

```
shinbuu@debian:~$ openssl s_server -accept 4433 -key key.pem -cert cert.pem -www
Using default temp DH parameters
ACCEPT
```

```
depth=0 CN=localhost
verify return:1
---
Certificate chain
 0 s:CN=localhost
  i:CN=localhost
   a:PKEY: RSA, 2048 (bit); sigalg: sha256WithRSAEncryption
   v:NotBefore: Oct  1 11:39:50 2025 GMT; NotAfter: Oct  1 11:39:50 2026 GMT
---
Server certificate
-----BEGIN CERTIFICATE-----
MIIDCTCCAFGgAwIBAgIUZg5PZxd4xobdglAQJUVIP8wwxi0wDQYJKoZIhvcNAQEL
3QAwFDESMBAGA1UEAwJbG9jYWxob3N0MB4XDTE1MTAwMTExMzYxMTFoXDTI2MTAw
MTExMzYxMTFoFDESMBAGA1UEAwJbG9jYWxob3N0MIIIBIjANBgkqhkiG9w0BAQEF
AAOCAQ8AMIIBCgKCAQEAqITL/KZJeFsQfMv0Cz5Ge0akhThrGXkSypurZsjs4+f
rpuyZIByFwe3kIgI3kjXiVZLJySV8axApCL2CQVmlmLZinVD84wRCBPwm2jXz7Yy
_7q093V9Sqb/btuVbvG3ljogTZ2DthcWsM9WjHIZBVhrvzr6BfS9NgEn+1GE3B0a
5ctcAA45WQC2voByBAi8G+r/SXPLeEnQkMYKGFVdt6UYjoAWOPtRyR0WipU3GL4
V0r5c6Y9EsbNlnHVPazo5x0lFhkZBB3m/KWjvKsf6csTnt+q1bXBvNkzCnmSTOSC
pBZMisCvjf6j50rn4Lho+Lq5o3o9ASd4Zb+/AS3vSQIDAQABO1MwUTAdBgNVHQ4E
FgQU/9NwqXjkteMwnIxYar1bkveUUh2QwHwYDVR0jBBgwFoAU/9NwqXjkteMwnIxY
ar1bkveUUh2QwHwYDVR0TAQH/BAUwAwEB/zANBgkqhkiG9w0BAQsFAA0CAQEAQK8
jXPAHvY685gPj5WxiRF3u3nFEI8rTAhaYVNIp3wV5bJAYebJiFJi1aSIQYcF8p9z
1ljZWVF8KA/QPpWAbr7xiGc7J710wT2ZbiS3nd/m/4YKzGqdsfxplwHv0vFiAzEW
32/KwEEW05nt43KehSeppf2kfEzbb1/QdlFWPy+xImtx7xvFAGhg27g0+Q00gEQq
Z605i0+BKVdcIGTa4VS40H0Leh40tY2YdqnmPY+Madfk+8iM6gvwzBPqeolsvYp
/1FUZCZFhTR7P/42qZsxVpx0bqgaSNxmru07fZ/uTtSkL9QPgNF5dcGYXBl7irpG
<IBLsMDiZcy8DfvAGQ==
-----END CERTIFICATE-----
subject=CN=localhost
issuer=CN=localhost
---
No client certificate CA names sent
Peer signing digest: SHA256
Peer signature type: rsa_pss_rsae_sha256
Negotiated TLS1.3 group: X25519MLKEM768
---
SSL handshake has read 2425 bytes and written 1558 bytes
```

## 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 encrypted 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
BDD8Z2Ef6YYnwICg2f3cDkJnYeqSC7HrFoUZ0Aw8RQA2ybeQngzjlZIFpUzr8l9g
90+hBgIEaNO3UqIEAgIcIKQGBAQBAAAArgcCBQC1RBPiswQCAHs
-----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:ECDSA+SHA512:ed25519:ed448:ecdsa_brainpoolP256r1_sha256:ecdsa_brainpoolP384r1_sha384:ecdsa_brainpoolP512r1_sha512:rsa_pss_pss_sha256:rsa_pss_pss_sha384:rsa_pss_pss_sha512:RSA-PSS+SHA256: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+SHA384:ECDSA+SHA512:ed25519:ed448:ecdsa_brainpoolP256r1_sha256:ecdsa_brainpoolP384r1_sha384:ecdsa_brainpoolP512r1_sha512:rsa_pss_pss_sha256:rsa_pss_pss_sha384:rsa_pss_pss_sha512:RSA-PSS+SHA256:RSA-PSS+SHA384:RSA-PSS+SHA512:RSA+SHA256:RSA+SHA384:RSA+SHA512
Supported groups: X25519MLKEM768:x25519:secp256r1:x448:secp384r1:secp521r1:ffdhe2048:ffdhe3072
Shared groups: X25519MLKEM768:x25519:secp256r1:x448:secp384r1:secp521r1:ffdhe2048:ffdhe3072
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.

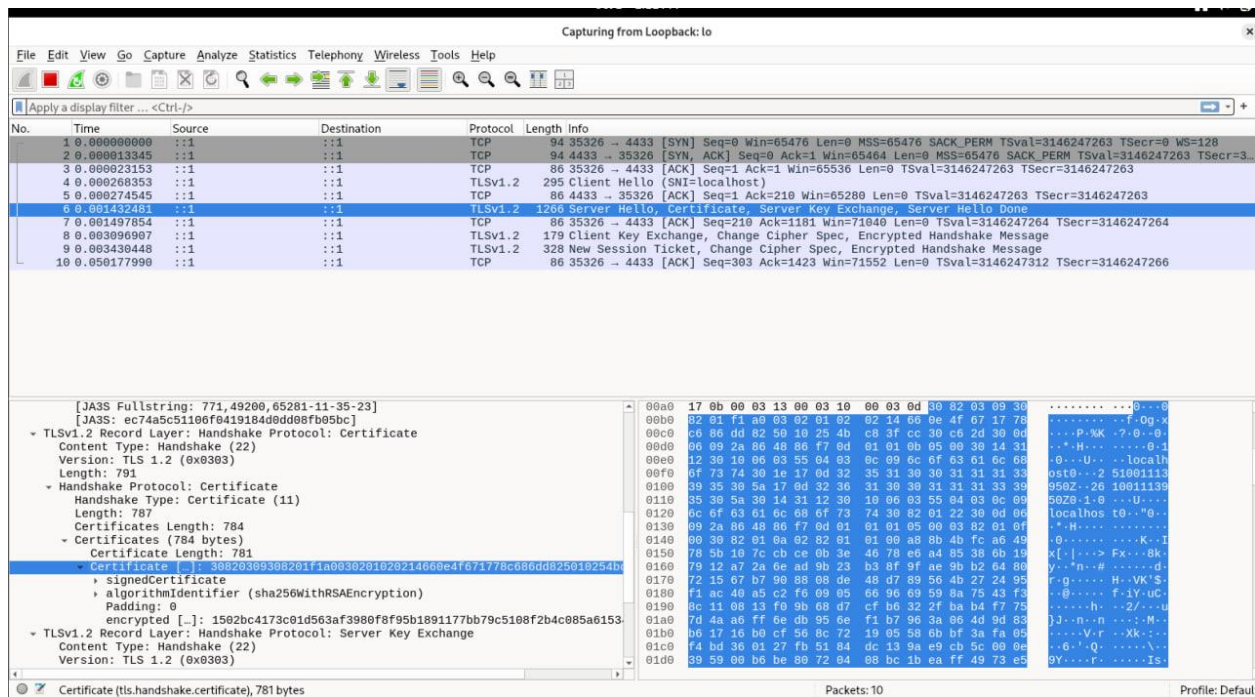
The image shows a Wireshark packet capture of a TLS 1.3 handshake. The packet list at the top shows the following packets:

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	:::	:::	TCP	94	58312 → 4433 [SYN] Seq=0 Win=65476 Len=0 MSS=65476 SACK_PERM TSval=2924377573 TSecr=0 WS=128
2	0.000000000	:::	:::	TCP	94	4433 → 58312 [SYN, ACK] Seq=0 Ack=1 Win=65464 Len=0 MSS=65476 SACK_PERM TSval=2924377573 TSecr=2
3	0.000000000	:::	:::	TCP	86	58312 → 4433 [ACK] Seq=1 Ack=1 Win=65536 Len=0 TSval=2924377573 TSecr=2924377573
4	0.000000000	:::	:::	TLSv1.3	1564	Client Hello (SNI=localhost)
5	0.000000000	:::	:::	TCP	86	4433 → 58312 [ACK] Seq=1 Ack=1479 Win=77952 Len=0 TSval=2924377574 TSecr=2924377574
6	0.000000000	:::	:::	TLSv1.3	2511	Server Hello, Change Cipher Spec, Application Data, Application Data, Application Data
7	0.000000000	:::	:::	TCP	86	58312 → 4433 [ACK] Seq=1479 Ack=2426 Win=91392 Len=0 TSval=2924377577 TSecr=2924377577
8	0.000000000	:::	:::	TLSv1.3	166	Change Cipher Spec, Application Data
9	0.000000000	:::	:::	TLSv1.3	341	Application Data
10	0.000000000	:::	:::	TLSv1.3	341	Application Data
11	0.000000000	:::	:::	TCP	86	58312 → 4433 [ACK] Seq=1559 Ack=2936 Win=92928 Len=0 TSval=2924377581 TSecr=2924377580

The packet details pane for the Client Hello (1) shows the following structure:

- Content Type: Handshake (22)
- Version: TLS 1.0 (0x0301)
- Length: 1473
- Handshake Type: Client Hello (1)
- Length: 1469
- Version: TLS 1.2 (0x0303)
- Random: fae66f1eb51a5a9be49c62718b30f985dbbf5fedb957c1ca57139b48d928
- Session ID Length: 32
- Session ID: ed6c9d1ccff88913d3d8be99cf6aafd5b12df2addce5fc2e8a5c7bb787fb8da4
- Cipher Suites (3 suites)
- Compression Methods Length: 1
- Compression Methods (1 method)
- Extensions Length: 1390
- Extension: server\_name (len=14) name=localhost
- Extension: ec\_point\_formats (len=4)
- Extension: supported\_versions (len=18) TLS 1.3
- Extension: session\_ticket (len=0)
- Extension: encrypt\_then\_mac (len=0)
- Extension: extended\_master\_secret (len=0)
- Extension: signature\_algorithms (len=42)
- Extension: supported\_versions (len=3) TLS 1.3
- Extension: psk\_key\_exchange\_modes (len=2)
- Extension: key\_share (len=1258) X25519MLKEM768, x25519
- Extension: compress\_certificate (len=5)
- [JA4: t13d931100\_55b375c5d22e\_199193a2bd39]





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:

```
shinbuu@debian: ~
40C789FD8C7F0000:error:0A000126:SSL routines:unexpected eof while reading:../ssl/record/rec_layer_s3.c:696:
shinbuu@debian:~$ openssl s_client -connect localhost:4433 -servername localhost -tls1_3
CONNECTED(00000003)
depth=0 CN=localhost
verify error:num=18:self-signed certificate
verify return:1
depth=0 CN=localhost
verify error:num=9:certificate is not yet valid
notBefore=Oct 7 00:00:00 2025 GMT
verify return:1
depth=0 CN=localhost
notBefore=Oct 7 00:00:00 2025 GMT
verify return:1
---
Certificate chain
 0 s:CN=localhost
 1 i:CN=localhost
  a:PKCS: RSA, 2048 (bit); sigalg: sha256WithRSAEncryption
  v:NotBefore: Oct 7 00:00:00 2025 GMT; NotAfter: Oct 8 00:00:00 2025 GMT
---
Server certificate
shinbuu@debian:~$ openssl s_client -connect localhost:4433 -servername localhost -tls1_3
4097F2263B7F0000:error:8000006F:system library:BIO_connect:Connection refused:../crypto/bio/bio_sock2.c:178:calling connect()
4097F2263B7F0000:error:10000067:BIO routines:BIO_connect:connect error:../crypto/bio/bio_sock2.c:180:
4097F2263B7F0000:error:8000006F:system library:BIO_connect:Connection refused:../crypto/bio/bio_sock2.c:178:calling connect()
4097F2263B7F0000:error:10000067:BIO routines:BIO_connect:connect error:../crypto/bio/bio_sock2.c:180:
connect:errno=111
shinbuu@debian:~$ openssl s_client -connect localhost:4433 -servername localhost -tls1_3
4047CAD1307F0000:error:8000006F:system library:BIO_connect:Connection refused:../crypto/bio/bio_sock2.c:178:calling connect()
4047CAD1307F0000:error:10000067:BIO routines:BIO_connect:connect error:../crypto/bio/bio_sock2.c:180:
4047CAD1307F0000:error:8000006F:system library:BIO_connect:Connection refused:../crypto/bio/bio_sock2.c:178:calling connect()
4047CAD1307F0000:error:10000067:BIO routines:BIO_connect:connect error:../crypto/bio/bio_sock2.c:180:
connect:errno=111
shinbuu@debian:~$ openssl s_server -accept 4433 -key key.pem -cert cert.pem -www
Using default temp DH parameters
ACCEPT
S
```

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.