CS528 Lab 2 Report
Zheng Zhong (zhong183)

Questions

Question 2.1 Think about why it is better to use UDP in the tunnel, instead of TCP, write your answer in your lab report.

Answer: Because VPN operates on the network (IP) layer, which is stateless and does not require the other features in TCP, e.g., reliable transmission, congestion control. These features should be implemented in upper layer applications that use the VPN. Thus using TCP connection for VPN is an over design.

Question 2.2 Think why it is not recommended to implement your own algorithm; and write that in your lab report.

Answer: Encryption and HMAC algorithms are a critical part of the VPN, which ensure confidentiality and integrity. Self-implemented algorithms can be error-prone. It's better to use stable and widely used implementations, such as OpenSSL, which are actively maintained by its community.

Question 2.3: Why is it important for the server to release resources when a connection is broken?

Answer: To save resources on the server side. Also, to reduce the attack surface.

Implementation

task 1
Modify simpletun.c to use UDP protocol, i.e., SOCK_STREAM -> SOCK_DGRAM, read() -> recvfrom(), write() -> sendto(). See `src/simpletun_udp_task1.c`.

task 2
Use OpenSSL implementation of AES256 and HMAC-SHA256 because they are among the strongest block ciphers and HMAC algorithms without effective attack today. see `src/minivpn_task2.c`, `src/openssl_utils.h`.

task 3-5
Use fork() to fork 2 processes after ssl certificate verification. Parent process handles control channel messages (e.g., close request from client and related handling at server) through ssl connection. Child process handles vpn traffic using UDP. The parent uses unnamed pipe to inform the child process to close connection and release resources. See `src/minivpn_task3_server.c`, `src/minivpn_task3_client.c`, `src/openssl_utils.h`.

task 6 and bonus

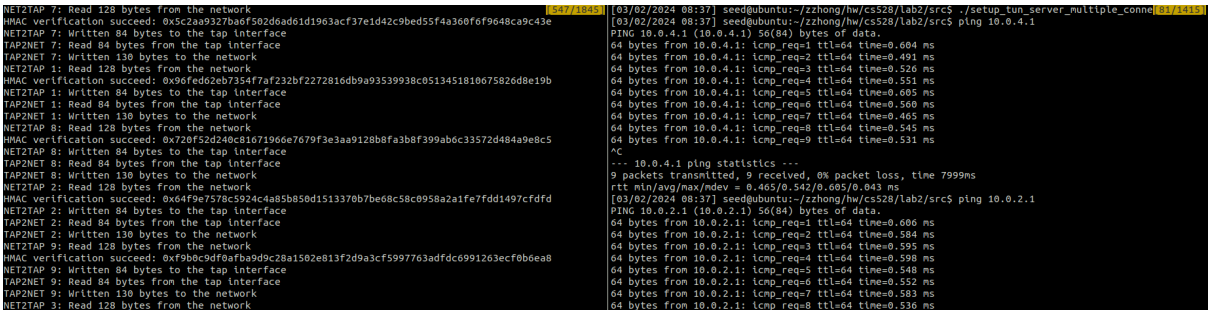
At the client side, use fork() to accept multiple connection requests. Upon receiving a new request, assign a new UDP port and send it back to the client using the ssl control channel. Bonus feature of session key and iv dynamic reconfiguration is implemented similar to the close request handling, i.e., handling the request through ssl control channel and use unnamed pipe to inform the child process. See `src/minivpn_task6_server.c`, `src/minivpn_task6_client.c`, `src/openssl_utils.h`.

Screenshots

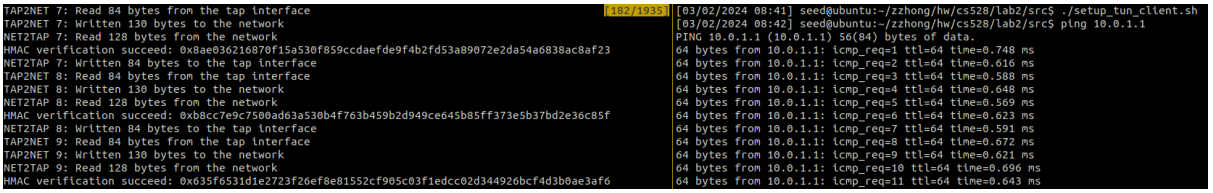
1. Multiple connections

Servers can accept multiple connection requests using ssl certificate for authentication; assign different UDP ports for different connections; exchange keys and ivs with clients separately; encrypt, decrypt and verify traffic through the VPN tunnels.

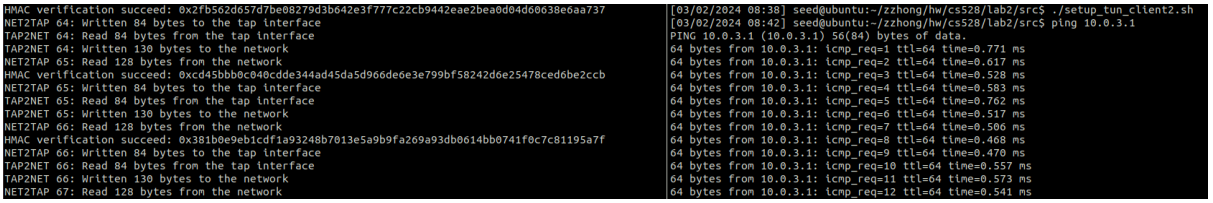
- server:



- client1:



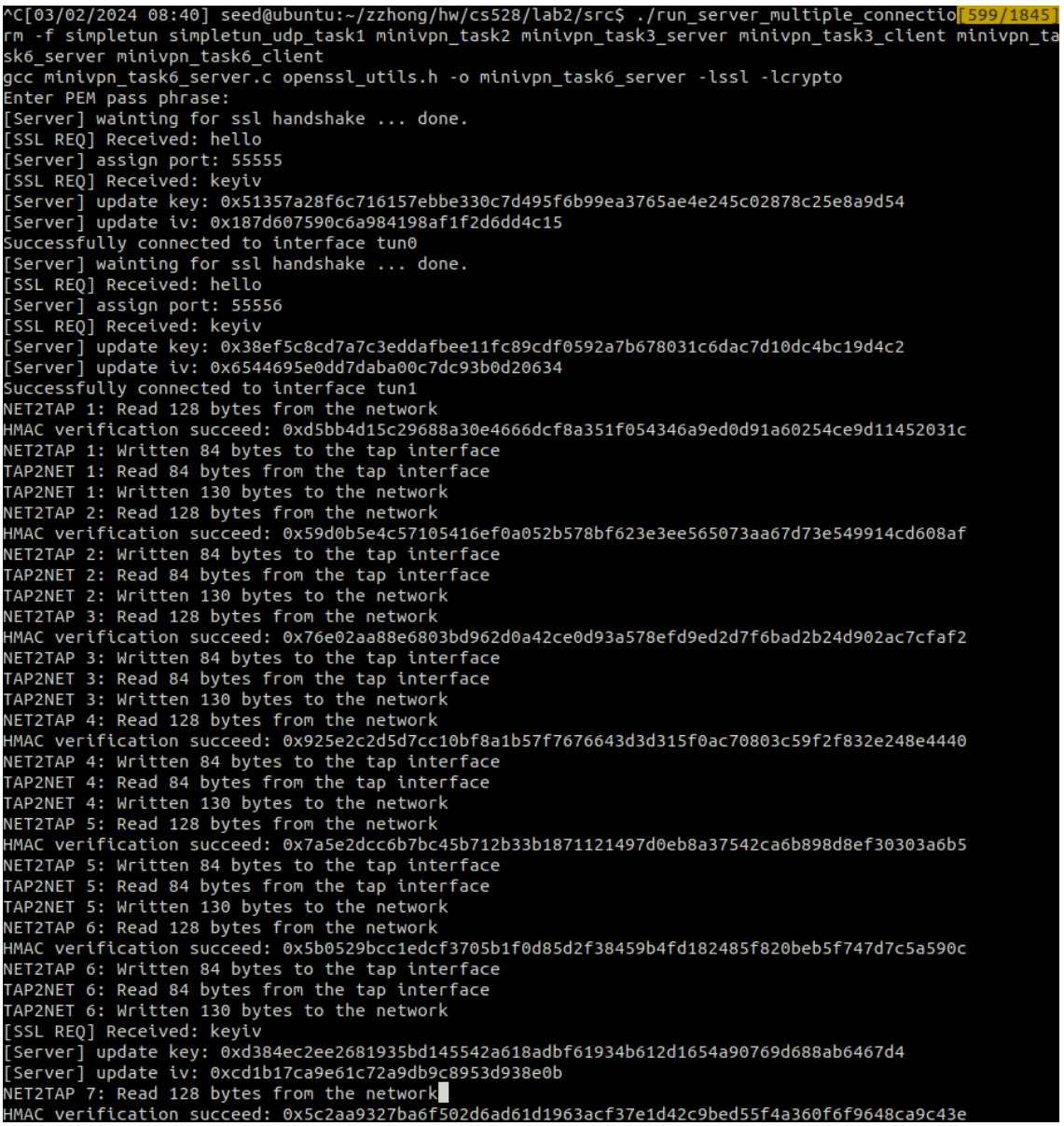
- client2:



2. Dynamic Reconfiguration (bonus feature)

Clients can send "keyiv" requests to the server and update the key and iv used.

- server:



- client:

```
^C[03/02/2024 08:40] seed@ubuntu:~/zzhong/hw/cs528/lab2/src5 ./run_client_multiple_connectio[228/1935]
rm -f simpletun simpletun_udp_task1 minivpn_task2 minivpn_task3_server minivpn_task3_client minivpn_ta
sk6_server minivpn_task6_client
gcc minivpn_task6_client.c openssl_utils.h -o minivpn_task6_client -lssl -lcrypto
Enter PEM pass phrase:
[Client] establishing tcp connection to server... done.
[Client] establishing ssl connection to server... done.
[SSL RES] Received: Hello from server!
[Client] obtain server assigned port: 55555
[SSL RES] Received: keyiv succeed
[Client] update key: 0x51357a28f6c716157ebbe330c7d495f6b99ea3765ae4e245c02878c25e8a9d54
[Client] update iv: 0x187d607590c6a984198af1f2d6dd4c15
Successfully connected to interface tun0
TAP2NET 1: Read 84 bytes from the tap interface
TAP2NET 1: Written 130 bytes to the network
NET2TAP 1: Read 128 bytes from the network
HMAC verification succeed: 0x8875c53053d6f83ce7ff24e7ed7ffef5f29640bdf92e2f2b587bdcafef3db5f1
NET2TAP 1: Written 84 bytes to the tap interface
TAP2NET 2: Read 84 bytes from the tap interface
TAP2NET 2: Written 130 bytes to the network
NET2TAP 2: Read 128 bytes from the network
HMAC verification succeed: 0xcd3ee497988990977e870179a20c12cc1288bbae3a15bc1b9c7ddae704217c8b
NET2TAP 2: Written 84 bytes to the tap interface
TAP2NET 3: Read 84 bytes from the tap interface
TAP2NET 3: Written 130 bytes to the network
NET2TAP 3: Read 128 bytes from the network
HMAC verification succeed: 0x3acf637da0c9f0648670d9aad6753c0ebd46279caf2ecc4a4a11745480421e07
NET2TAP 3: Written 84 bytes to the tap interface
TAP2NET 4: Read 84 bytes from the tap interface
TAP2NET 4: Written 130 bytes to the network
NET2TAP 4: Read 128 bytes from the network
HMAC verification succeed: 0x16726aae33d478930ceca7f93e775c1d711ac979acfeff7960a16261476cadbe
NET2TAP 4: Written 84 bytes to the tap interface
TAP2NET 5: Read 84 bytes from the tap interface
TAP2NET 5: Written 130 bytes to the network
NET2TAP 5: Read 128 bytes from the network
HMAC verification succeed: 0xa1d06e7843d0025932b2a20ff1b1266366626c5c3123ac8049933717ea19a36f
NET2TAP 5: Written 84 bytes to the tap interface
TAP2NET 6: Read 84 bytes from the tap interface
TAP2NET 6: Written 130 bytes to the network
NET2TAP 6: Read 128 bytes from the network
HMAC verification succeed: 0x5a0d2a39613b15d8c978f805fdd89774af86be516efae5c5ae61022609e34ab5
NET2TAP 6: Written 84 bytes to the tap interface
[SSL RES] Received: keyiv succeed
[Client] update key: 0xd384ec2ee2681935bd145542a618adbf61934b612d1654a90769d688ab6467d4
[Client] update iv: 0xcd1b17ca9e61c72a9db9c8953d938e0b
TAP2NET 7: Read 84 bytes from the tap interface
TAP2NET 7: Written 130 bytes to the network
NET2TAP 7: Read 128 bytes from the network
HMAC verification succeed: 0x8ae036216870f15a530f859ccdaefde9f4b2fd53a89072e2da54a6838ac8af23
NET2TAP 7: Written 84 bytes to the tap interface
TAP2NET 8: Read 84 bytes from the tap interface
TAP2NET 8: Written 130 bytes to the network
NET2TAP 8: Read 128 bytes from the network
```