CSBU111.P11.KHBC LAB REPORT

23560026 – Vũ Đăng khương

Contents

[LAB 1 3](#_Toc187052123)

[Setup 3](#_Toc187052124)

[GCC 3](#_Toc187052125)

[G++ 3](#_Toc187052126)

[Clang 3](#_Toc187052127)

[Cl 3](#_Toc187052128)

[NASM 3](#_Toc187052129)

[OpenSSL Librariees for GCC, Clang and MSVC 4](#_Toc187052130)

[Test compiling with OpenSSL Libraries 5](#_Toc187052131)

[GCC 5](#_Toc187052132)

[Clang 6](#_Toc187052133)

[MSVC 7](#_Toc187052134)

[Encryption algorithms functions explanation 8](#_Toc187052135)

[AES 8](#_Toc187052136)

[RSA 9](#_Toc187052137)

[LAB 2 10](#_Toc187052138)

[Setup Apache 10](#_Toc187052139)

[Setup Apache on Windows 10](#_Toc187052140)

[Testing and Setting Up Apache 10](#_Toc187052141)

[Create Digital Certificate and Enables HTTPS 11](#_Toc187052142)

[Verify SSL with OPENSSL 12](#_Toc187052143)

[LAB 3 13](#_Toc187052144)

[Setup OpenVPN 13](#_Toc187052145)

[Server 13](#_Toc187052146)

[Client 16](#_Toc187052147)

[Running 18](#_Toc187052148)

[LAB 4 20](#_Toc187052149)

[Setup UFW 20](#_Toc187052150)

[Rules implemeted: 20](#_Toc187052151)

[LAB 5 21](#_Toc187052152)

[Setup SSH 21](#_Toc187052153)

[Installing SSH 21](#_Toc187052154)

[Check SSH 22](#_Toc187052155)

[Remote login 22](#_Toc187052156)

[Setup Iptables 22](#_Toc187052157)

# LAB 1

## Setup

### GCC

A black background with white text

Description automatically generated

### G++

A black background with white text

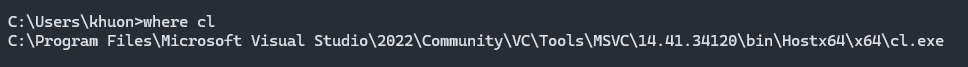
Description automatically generated

### Clang

A black background with white text

Description automatically generated

### Cl



### NASM

A close up of a logo

Description automatically generated

### OpenSSL Librariees for GCC, Clang and MSVC

A screenshot of a computer

Description automatically generated

* GCC:

A screenshot of a computer

Description automatically generated

* Clang:

A screenshot of a computer

Description automatically generated

* MSVC:

A screenshot of a computer

Description automatically generated

## Test compiling with OpenSSL Libraries

* Compiled:

A screenshot of a black screen

Description automatically generated

### GCC

* Configure:

A screen shot of a computer program

Description automatically generated

* Run:

A black background with white text

Description automatically generated

### Clang

* Configure:

A computer screen shot of text

Description automatically generated

* Run:

A computer code on a black background

Description automatically generated

### MSVC

* Configure:

A screenshot of a computer program

Description automatically generated

* Run:

A computer code on a black background

Description automatically generated

## Encryption algorithms functions explanation

### AES

* **print\_openssl\_errors()**  
  Outputs OpenSSL error messages to the standard error stream. This function is useful for troubleshooting OpenSSL-related issues by printing the error details to stderr.
* **hex\_to\_bytes(const std::string& hex)**  
  Converts a hexadecimal string (e.g., "aabbcc") into a vector of bytes (unsigned char). It processes the string in pairs of characters, converts each pair into a byte, and appends it to the resulting vector.
* **generate\_key\_iv(std::vector<unsigned char>& key, std::vector<unsigned char>& iv, int key\_size)**  
  Creates a random AES key and initialization vector (IV) using OpenSSL's RAND\_bytes() function. The key size (in bytes) is specified as an argument, while the IV size is set to 16 bytes (128 bits). The function adjusts the size of the key and IV vectors and fills them with random data.
* **aes\_encrypt(const std::vector<unsigned char>& plaintext, std::vector<unsigned char>& ciphertext, const std::vector<unsigned char>& key, const std::vector<unsigned char>& iv, const EVP\_CIPHER\* cipher)**  
  Encrypts the given plaintext using AES in the specified mode (e.g., ECB, CBC). It sets up an encryption context with EVP\_EncryptInit\_ex(), processes the data with EVP\_EncryptUpdate(), and finalizes encryption using EVP\_EncryptFinal\_ex(). The resulting ciphertext is stored in the output vector.
* **aes\_decrypt(const std::vector<unsigned char>& ciphertext, std::vector<unsigned char>& plaintext, const std::vector<unsigned char>& key, const std::vector<unsigned char>& iv, const EVP\_CIPHER\* cipher)**  
  Decrypts the provided ciphertext using AES in the specified mode. It initializes a decryption context with EVP\_DecryptInit\_ex(), processes the data using EVP\_DecryptUpdate(), and finalizes decryption with EVP\_DecryptFinal\_ex(). The resulting plaintext is stored in the output vector.
* **get\_cipher(const std::string& mode)**  
  Returns the appropriate EVP\_CIPHER structure from OpenSSL for a given mode (e.g., "ecb", "cbc", "ctr"). It maps the mode string to a corresponding AES cipher (e.g., EVP\_aes\_256\_ecb(), EVP\_aes\_256\_cbc()). If the mode is invalid, it prints an error message and terminates the program.
* **main(int argc, char\* argv[])**  
  Serves as the program's entry point, handling command-line arguments. It expects the AES cipher mode and operation type (e.g., "encrypt", "decrypt") as inputs. Depending on the operation, it generates a key and IV or performs encryption or decryption. The function processes inputs and executes the appropriate actions based on user arguments.

### RSA

* **print\_openssl\_errors()**

Outputs any OpenSSL error messages to the standard error stream (stderr). This is helpful for debugging OpenSSL-related issues.

* **generate\_rsa\_keys(int bits, const std::string& public\_key\_file, const std::string& private\_key\_file)**

Creates an RSA key pair with the specified bit size (e.g., 2048 or 4096). The public key is saved in the given public key file, and the private key is stored in the specified private key file, both in OpenSSL's PEM format.

* **load\_public\_key(const std::string& public\_key\_file)**

Reads an RSA public key from a PEM-formatted file. It opens the file and retrieves the key using OpenSSL’s PEM\_read\_bio\_RSA\_PUBKEY() function.

* **load\_private\_key(const std::string& private\_key\_file)**

Reads an RSA private key from a PEM-formatted file. It opens the file and loads the key using OpenSSL’s PEM\_read\_bio\_RSAPrivateKey() function.

* **rsa\_encrypt(RSA\* rsa, const std::vector<unsigned char>& plaintext)**

Encrypts the provided plaintext using the specified RSA public key. The function uses RSA\_public\_encrypt() with RSA\_PKCS1\_OAEP\_PADDING to generate the ciphertext.

* **bio\_read(const std::string& filename)**

Reads file contents using OpenSSL’s BIO functions. The function processes data in 1024-byte chunks and stores it in a vector of unsigned characters, which it returns. This is commonly used for reading input files for encryption or decryption.

* **bio\_write(const std::string& filename, const std::vector<unsigned char>& data)**

Writes data to a file using OpenSSL’s BIO functions. The BIO\_write() function writes the byte data to the output file, useful for saving encrypted or decrypted files.

* **main(int argc, char\* argv[])**

The main function handles command-line arguments to perform tasks such as generating RSA keys, encrypting files with the public key, or decrypting files with the private key.

# LAB 2

## Setup Apache

### Setup Apache on Windows

A screenshot of a computer

Description automatically generated

* After installing Apache and move it to C:/ directory, the PC C:/ Directory now looks like this

### Testing and Setting Up Apache

A screenshot of a computer

Description automatically generated

* After installing when access via localhost we can see the website being displayed

A screenshot of a computer

Description automatically generated

* After httpd –k install the service has been added to the services list of the machine

### Create Digital Certificate and Enables HTTPS

* After the register a domain, we create an ECC Private key and CSR with OPENSSL using
  + openssl ecparam -genkey -name prime256v1 -out ec-private-key.pem
  + openssl req -new -key ec-private-key.pem -out ec-request.csr

**A screenshot of a computer program

Description automatically generated**

* After the certificate has been issued, it gets downloaded and put into C:/Apache24/ssl. Noted that the certificate is signed by ZeroSSL

**A computer screen with white text

Description automatically generated**

### Verify SSL with OPENSSL

* After the completion of setting up TLS for the Apache server, when verifying with openssl we get

A screenshot of a computer

Description automatically generated

* And when accessing it via browser (after changinf the hosts file) we can see that the Apache server is now using https

A screenshot of a computer

Description automatically generated

# LAB 3

## Setup OpenVPN

### Server

A screen shot of a computer

Description automatically generated

**1. Protocol and Port**

proto tcp

port 1194

* **proto tcp**: Specifies that the server uses the TCP protocol for communication. OpenVPN also supports UDP (proto udp), which can provide better performance for latency-sensitive applications.
* **port 1194**: The server listens on port 1194, the default port for OpenVPN traffic.

**2. Device Mode**

dev tun2

* **dev tun2**: Configures the server to use a TUN (tunneling) device named tun2. TUN devices handle Layer 3 traffic (IP packets) and are commonly used for routing.

**3. Cryptographic Configuration**

ca /etc/openvpn/ca.crt

cert /etc/openvpn/server.crt

key /etc/openvpn/server.key

key-direction 1

dh none

tls-groups prime256v1

cipher AES-256-GCM

auth SHA512

data-ciphers AES-256-GCM:AES-128-GCM:?CHACHA20-POLY1305

* **ca**, **cert**, **key**: File paths to the CA (Certificate Authority) certificate, server certificate, and private key, which authenticate the server to clients.
* **key-direction 1**: Sets the direction for the TLS authentication key if used.
* **dh none**: Disables Diffie-Hellman parameters since elliptic curve cryptography is used.
* **tls-groups prime256v1**: Specifies the elliptic curve (prime256v1) for secure key exchange.
* **cipher AES-256-GCM**: Defines AES-256-GCM as the primary encryption cipher for secure communication.
* **auth SHA512**: Sets HMAC (Hash Message Authentication Code) to SHA-512 for integrity checks.
* **data-ciphers**: Lists supported ciphers for negotiation with clients, including optional ChaCha20-Poly1305.

**4. Network Settings**

server 10.8.0.0 255.255.255.0

push "redirect-gateway def1 bypass-dhcp"

push "dhcp-option DNS 8.8.8.8"

push "dhcp-option DNS 8.8.4.4"

topology subnet

* **server 10.8.0.0 255.255.255.0**: Assigns the 10.8.0.0/24 subnet to VPN clients.
* **push "redirect-gateway def1 bypass-dhcp"**: Routes all client traffic through the VPN to ensure secure internet access.
* **push "dhcp-option DNS 8.8.8.8"**, **push "dhcp-option DNS 8.8.4.4"**: Configures Google DNS servers for client name resolution.
* **topology subnet**: Specifies a subnet topology where each client gets a unique IP address.

**5. Security Settings**

user nobody

group nogroup

persist-key

persist-tun

explicit-exit-notify 1

* **user nobody** and **group nogroup**: Runs the server process with non-root privileges to enhance security.
* **persist-key**: Prevents the private key from being reloaded on restarts.
* **persist-tun**: Keeps the TUN device active during server restarts.
* **explicit-exit-notify 1**: Sends a notification to clients when the server shuts down.

**6. Logging and Verbosity**

status /var/log/openvpn-status.log

log /var/log/openvpn.log

log-append /var/log/openvpn.log

verb 5

* **status**: Specifies the path for real-time status logs.
* **log**: Defines the main log file path for server activity.
* **log-append**: Appends new logs to the specified file.
* **verb 5**: Sets the log verbosity level to 5, providing detailed information for debugging.

**8. Keepalive to Ensure Connection Stability**

keepalive 10 120

* **keepalive 10 120**: Sends a ping every 10 seconds and expects a response within 120 seconds to maintain the connection.

### Client

A screenshot of a computer program

Description automatically generated

**1. Client Mode**

client

* **client**: Configures the instance to operate in client mode.

**2. VPN Device**

dev tun2

* **dev tun2**: Specifies the use of a TUN device named tun2, which handles Layer 3 traffic.

**3. Protocol and Server Address**

proto tcp

remote 10.8.0.1 1194

* **proto tcp**: Indicates that the client uses TCP to communicate with the server.
* **remote 10.8.0.1 1194**: Specifies the server's IP address (replace with your server's public IP or domain) and port.

**4. Connection Options**

resolv-retry infinite

nobind

persist-key

persist-tun

* **resolv-retry infinite**: Retries the connection indefinitely if it is lost.
* **nobind**: Prevents binding to a specific local port.
* **persist-key**: Avoids reloading the key across restarts.
* **persist-tun**: Maintains the TUN device's state across restarts.

**5. Cryptographic Settings**

ca /etc/openvpn/client/ca.crt

cert /etc/openvpn/client/client.crt

key /etc/openvpn/client/client.key

key-direction 1

cipher AES-256-GCM

data-ciphers AES-256-GCM:AES-128-GCM:?CHACHA20-POLY1305

tls-groups prime256v1

auth SHA512

* **ca**, **cert**, **key**: File paths to the CA certificate, client certificate, and private key.
* **key-direction 1**: Matches the key direction defined in the server configuration.
* **cipher AES-256-GCM**: Encryption cipher for secure communication.
* **data-ciphers**: Specifies supported ciphers for negotiation.
* **tls-groups prime256v1**: Matches the server's elliptic curve group.
* **auth SHA512**: Configures HMAC for data integrity verification.

**6. Redirect Traffic and DNS Settings**

redirect-gateway def1

dhcp-option DNS 8.8.8.8

dhcp-option DNS 8.8.4.4

* **redirect-gateway def1**: Routes all traffic through the VPN.
* **dhcp-option DNS**: Configures Google DNS servers for the client.

**7. Logging and Verbosity**

verb 5

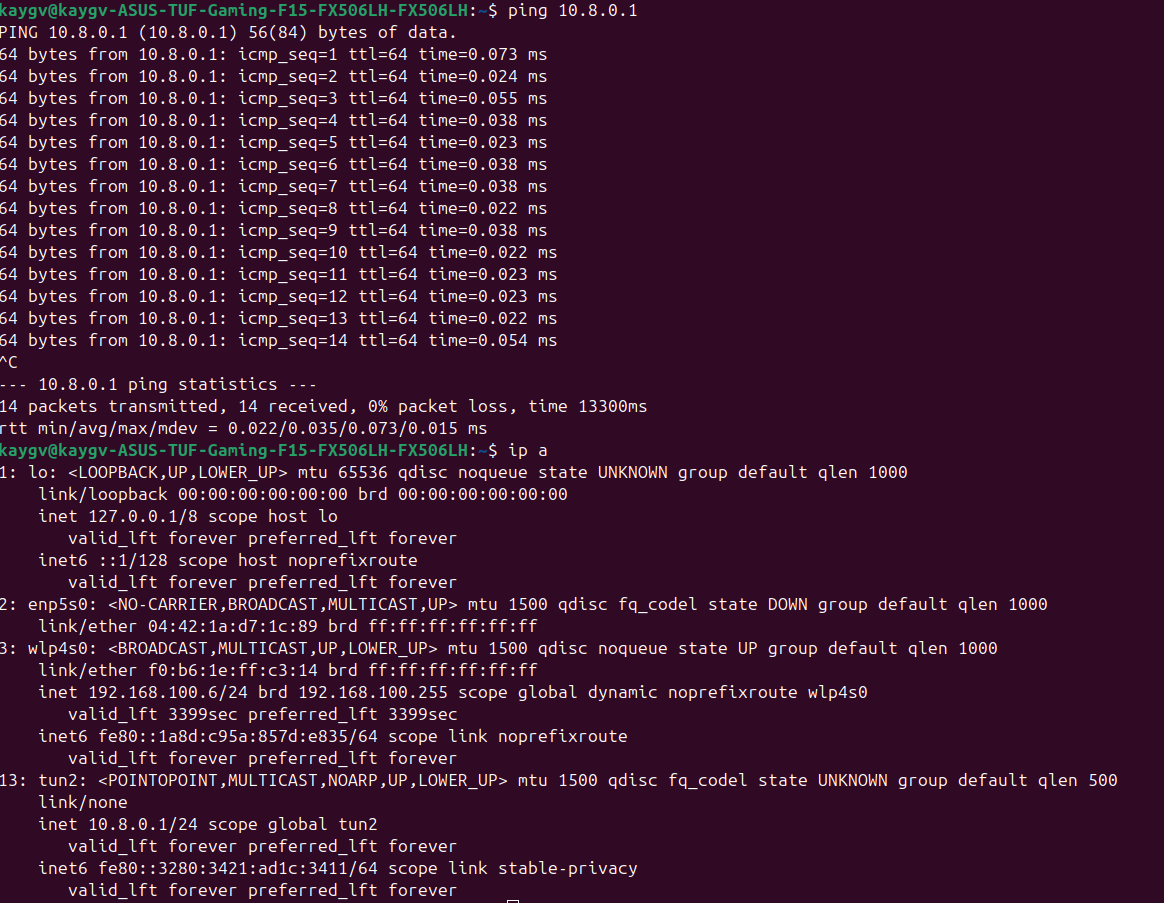
log /var/log/openvpn-client.log

* **verb 5**: Sets log verbosity for detailed debugging.
* **log**: Specifies the client log file path.

## Running



After running successfully, the tun2 appears in ip route and we can ping the vpn server



# LAB 4

## Setup UFW

A screenshot of a computer program

Description automatically generated

### Rules implemeted:

#### Incoming Rules:

1. **1194/udp and 1194/tcp (ALLOW Anywhere):**
   * UDP and TCP traffic on port 1194 is allowed
2. **OpenSSH (ALLOW Anywhere):**
   * Allows SSH connections. This is for remote server management.
3. **Nginx Full (ALLOW Anywhere):**
   * Permits HTTP (port 80) and HTTPS (port 443) traffic for web servers running Nginx.
4. **22 (LIMIT Anywhere):**
   * Limits incoming SSH (port 22) attempts to prevent brute-force attacks. Excessive connections will be blocked temporarily.
5. **DENY 192.168.1.100 and 192.168.1.0/24:**
   * Denies traffic from the IP address 192.168.1.100 and the entire subnet 192.168.1.0/24 (local network).
6. **443 (ALLOW Anywhere):**
   * Allows incoming HTTPS traffic.
7. **80 (ALLOW Anywhere):**
   * Allows incoming HTTP traffic.

#### Outgoing Rules:

1. **80 (ALLOW OUT Anywhere):**
   * Outbound HTTP traffic (port 80) is allowed.
2. **443 (ALLOW OUT Anywhere):**
   * Outbound HTTPS traffic (port 443) is allowed.
3. **53 (ALLOW OUT Anywhere):**
   * Outbound DNS traffic (port 53) is allowed. This is needed for domain name resolution.
4. **IPv6 OUT Rules:**
   * Outbound traffic rules for HTTP, HTTPS, and DNS also apply to IPv6.

#### Forwarding Rules:

1. **Anywhere on eth0 (ALLOW FWD):**
   * Allows traffic forwarding between eth0 (a network interface) and other destinations.
2. **Anywhere on tun0 (ALLOW FWD):**
   * Allows traffic forwarding between tun0 (likely a VPN interface) and other destinations.

# LAB 5

## Setup SSH

### Installing SSH



### Check SSH

A computer screen shot of a program

Description automatically generated

## Remote login

A screenshot of a computer program

Description automatically generated

* After setup we can now login remotely using both password and certificate

## Setup Iptables

