

HỌC VIỆN NGÂN HÀNG
KHOA HỆ THỐNG THÔNG TIN QUẢN LÝ



BÁO CÁO BÀI TẬP LỚN
AN TOÀN VÀ BẢO MẬT THÔNG TIN
ĐỀ TÀI: Tiêu chuẩn Việt Nam về mã hoá AES
và thư viện phần mềm OpenSSL

Giảng viên hướng dẫn : TS.Vũ Duy Hiến

Lớp : K24CNTTA

Nhóm : 5

Hà Nội – 2023

HỌC VIỆN NGÂN HÀNG
KHOA HỆ THỐNG THÔNG TIN QUẢN LÝ



BÁO CÁO BÀI TẬP LỚN
AN TOÀN VÀ BẢO MẬT THÔNG TIN
ĐỀ TÀI: Tiêu chuẩn Việt Nam về mã hoá AES
và thư viện phần mềm OpenSSL

Thành viên	Mã sinh viên	Tỉ lệ đóng góp	Xác nhận
Đinh Thành Nam (Nhóm trưởng)	24A4042599	22%	
Nguyễn Duy Hưng	24A4042596	21%	
Trần A Khang	24A4040141	21%	
Trương Ngọc Minh Khôi	24A4040167	15%	
Nguyễn Thế Nghĩa	24A4042602	21%	

Hà Nội – 2023

LỜI CẢM ƠN

Lời đầu tiên, nhóm 5 xin gửi lời cảm ơn đến Học viện Ngân hàng và khoa Hệ thống thông tin quản lý đã tạo điều kiện cho chúng em và các bạn sinh viên có cơ hội được học tập trong một môi trường năng động, sáng tạo, tạo điều kiện nỗ lực và phát huy hết tiềm năng của mình. Nhóm xin gửi lời cảm ơn đặc biệt đến thầy Vũ Duy Hiến đã dành sự tâm huyết và nhiệt tình giảng dạy, giúp chúng em học được một môn học có ứng dụng cao trong ngành học và nghiệp vụ sau này, đáp ứng nhu cầu thực tiễn của các nhà tuyển dụng về kiến thức, kỹ năng cần có.

Tiếp nối các học phần trước đó, học phần này giúp sinh viên tiếp tục tìm hiểu, học tập và nghiên cứu các công cụ thực tế mà doanh nghiệp sử dụng, trong lĩnh vực công nghệ thông tin, đặc biệt là lĩnh vực an toàn và bảo mật. Song vì chưa có nhiều kinh nghiệm nên bài báo cáo sẽ không tránh khỏi những thiếu sót, kính mong thầy nhận xét, góp ý để bài báo cáo của chúng em được hoàn thiện hơn, rút kinh nghiệm cho các dự án tiếp theo.

Chúng em xin chân thành cảm ơn!

Nhóm sinh viên thực hiện

Nhóm 5

LỜI CAM ĐOAN

Nhóm 5 xin cam đoan báo cáo này là sản phẩm nghiên cứu và thực hành của nhóm. Bài báo cáo đảm bảo tính liêm chính trong học tập, không đạo văn, gian lận, bịa đặt. Các thông tin tham khảo được trích dẫn nguồn đầy đủ và minh bạch.

Nhóm 5 xin chịu toàn bộ trách nhiệm nếu bài báo cáo vi phạm các điều trên.

Hà Nội, ngày 12 tháng 5 năm 2024

Đại diện nhóm 5

Đinh Thành Nam

MỤC LỤC

LỜI CẢM ƠN	1
LỜI CAM ĐOAN	2
MỤC LỤC	3
DANH MỤC HÌNH ẢNH	5
CHƯƠNG I: TIÊU CHUẨN VIỆT NAM VỀ MÃ HOÁ AES	7
1. Tổng quan về AES	7
2. Tiêu chuẩn Việt Nam về mã khóa đối xứng AES	7
3. Thuật toán của AES.....	7
4. Mô tả thuật toán AES bằng mã nguồn C++	11
CHƯƠNG II: GIỚI THIỆU VỀ OPENSSL	25
1. Giới thiệu:	25
2. Lịch sử:	25
3. Ưu và nhược điểm:	25
3.1. Ưu điểm:	25
3.2. Nhược điểm:	26
CHƯƠNG III: XÂY DỰNG, CÀI ĐẶT VÀ LÀM VIỆC VỚI OPENSSL	28
1. Xây dựng và cài đặt OpenSSL cho Windows 10 x64	28
1.1 Tải về mã nguồn của OpenSSL	29
1.2. Giải nén	29
1.3. Sử dụng Perl để configure trong Visual Studio Command Prompt	30
1.4. Sử dụng nmake để biên dịch mã nguồn của OpenSSL	32
1.5. Tích hợp OpenSSL vào phát triển chương trình C dùng Visual Studio Code	34
1.6. Làm việc với nhiều file mã nguồn và sử dụng debug & run trong VS Code	37
2. Xây dựng và cài đặt OpenSSL cho Linux (Ubuntu)	40
2.1. Kiểm tra phiên bản OpenSSL	40
2.2. Tải và giải nén mã nguồn của OpenSSL	40

2.3. Chạy file cấu hình bằng lệnh ./config.....	41
2.4. Biên dịch và cài đặt OpenSSL	42
2.5. Cấu hình để sử dụng phiên bản OpenSSL mới nhất và thư viện của nó	42
2.6. Làm việc với ngôn ngữ C trên Linux có sử dụng thư viện của OpenSSL	44
ĐIỀU KIỆN TIỀN QUYẾT	46
1. Strawberry Perl	46
1.1. Giới thiệu	46
1.2. Lý do cần thiết	46
1.3. Tải và cài đặt	47
2. NASM (Netwide Assembler)	48
2.1. Giới thiệu	48
2.2. Lý do cần thiết	48
2.3. Tải và cài đặt	49
3. Cài đặt gói phát triển với C++ của Visual Studio	51
4. MSYS2	53
4.1. Giới thiệu	53
4.2. Mục đích và tính năng	53
4.3. Cài đặt và sử dụng	54
5. Windows subsystem for linux (WSL)	56
5.1. Giới thiệu	56
5.2. Cài đặt.....	56
5. Kiểm tra biến môi trường	59
KẾT LUẬN	62
Tài liệu tham khảo.....	63

DANH MỤC HÌNH ẢNH

Hình 1: Mô hình hoá thuật toán AES	9
Hình 2: Hàm ShiftRows	10
Hình 3: Hàm MixColumn	10
Hình 4: Mô hình hoá giai đoạn mở rộng khoá	10
Hình 5: Logo OpenSSL	25
Hình 6: Giải nén mã nguồn OpenSSL	29
Hình 7: x64 Native Tools Command Prompt for VS 2022	30
Hình 8: Cấu hình trước khi cài đặt OpenSSL bằng Perl configure	31
Hình 9: Biên dịch mã nguồn OpenSSL trên Windows	32
Hình 10: Test trước khi cài đặt OpenSSL trên Windows	32
Hình 11: Cài đặt OpenSSL trên Windows	33
Hình 12: Cấu hình đường dẫn biến môi trường cho OpenSSL	33
Hình 13: Làm việc với C/C++ bằng Visual Studio Code	34
Hình 14: Extension để làm việc với C/C++	35
Hình 15: Cấu hình để làm việc với C/C++	35
Hình 16: Cấu hình compiler path và include path	35
Hình 17: Biên dịch và chạy bằng PowerShell	36
Hình 18: Biên dịch và chạy bằng Command Prompt	36
Hình 19: Cấu hình file c_cpp_properties.json	37
Hình 20: Cấu hình file launch.json	37
Hình 21: Cấu hình tasks.json	38
Hình 22: Kiểm tra OpenSSL trên Linux	40
Hình 23: Giải nén mã nguồn của OpenSSL	41
Hình 24: Tải xuống mã nguồn của OpenSSL	41
Hình 25: Cấu hình trước khi biên dịch và cài đặt OpenSSL	41
Hình 26: Biên dịch mã nguồn OpenSSL trên Linux	42
Hình 27: Cấu hình cho Linux sử dụng bản OpenSSL mong muốn(1)	43
Hình 28: Kiểm tra phiên bản OpenSSL đang được sử dụng trên Linux	43
Hình 29: Cấu hình cho Linux sử dụng bản OpenSSL mong muốn(2)	43
Hình 30: Cấu hình cho Linux sử dụng bản OpenSSL mong muốn(3)	43
Hình 31: Kiểm tra lại phiên bản OpenSSL sau khi đã thiết lập lại	44
Hình 32: Cài đặt WSL cho Visual Studio Code	44

Hình 33: Sử dụng apt-get để cài đặt trên Linux	45
Hình 34: Giao diện bash Ubuntu tích hợp trong Visual Studio Code	45
Hình 35: Logo Strawberry Perl	46
Hình 36: Tải và cài đặt Strawberry Perl trên Windows	47
Hình 37: Kiểm tra cài đặt thành công Perl	47
Hình 38: Logo NASM	48
Hình 39: Tải và cài đặt NASM (1).....	49
Hình 40: Tải và cài đặt NASM (2).....	50
Hình 41: Tải và cài đặt NASM (3).....	50
Hình 42: Cài đặt gói phát triển với C++ của Visual Studio (1)	51
Hình 43: Cài đặt gói phát triển với C++ của Visual Studio (2)	51
Hình 44: logo MSYS2	53
Hình 45: Tải và cài đặt MSYS2	54
Hình 46: Tải gcc và gdb	54
Hình 47: Thêm thư viện UCRT64 vào biến môi trường	55
Hình 48: Logo WSL	56
Hình 49: Kiểm tra phiên bản Windows	57
Hình 50: Kiểm tra cấu hình ảo hoá CPU (Virtualization)	57
Hình 51: Cài đặt WSL	58
Hình 52: Cấu hình username và password cho distro	58
Hình 53: Kiểm tra và thêm biến môi trường (1).....	59
Hình 54: Kiểm tra và thêm biến môi trường (2).....	59
Hình 55: Kiểm tra và thêm biến môi trường (3).....	60
Hình 56: Kiểm tra và thêm biến môi trường (4).....	60
Hình 57: Kiểm tra và thêm biến môi trường (5).....	61
Hình 58: Kiểm tra và thêm biến môi trường (6).....	61

CHƯƠNG I: TIÊU CHUẨN VIỆT NAM VỀ MÃ HOÁ AES

1. Tổng quan về AES

AES (Advanced Encryption Standard), một tiêu chuẩn mã hóa khối được chính phủ Hoa Kỳ áp dụng, là sản phẩm của công việc phát triển từ **Rijndael Cipher** của hai nhà mật mã học người Bỉ, **Joan Daemen** và **Vincent Rijmen**. Thuật toán này hoạt động trên các khối dữ liệu có độ dài 128 bit và sử dụng các khóa có độ dài 128 bit, 192 bit hoặc 256 bit. Quá trình mở rộng khóa trong chu trình mã hóa được thực hiện bằng thủ tục sinh khóa Rijndael. **AES** thực hiện hầu hết các phép toán trong một trường hữu hạn của các byte. Mỗi khối dữ liệu 128 bit được chia thành 16 byte, có thể được sắp xếp thành một ma trận 4x4 của các byte, gọi là ma trận trạng thái. Số lần lặp trong thuật toán phụ thuộc vào độ dài của khóa, có thể là 128 bit, 192 bit hoặc 256 bit.

2. Tiêu chuẩn Việt Nam về mã khóa đối xứng AES

Tiêu chuẩn **AES** là một trong những tiêu chuẩn mã hóa đối xứng phổ biến nhất trên toàn thế giới, bao gồm cả Việt Nam. Nó đã trở thành một phần không thể thiếu trong các hệ thống và ứng dụng bảo mật thông tin, từ việc bảo vệ dữ liệu cá nhân đến giao tiếp an toàn trên internet và nhiều lĩnh vực khác. Việt Nam đã công bố thuật toán **AES** thành tiêu chuẩn quốc gia **TCVN 7816:2007** năm **2007** về Thuật toán mã hóa dữ liệu **AES**.

Trong ngữ cảnh của Việt Nam, việc sử dụng **AES** phản ánh sự phát triển và ứng dụng của công nghệ thông tin và bảo mật thông tin trong xã hội hiện đại. Các tổ chức, doanh nghiệp và cơ quan chính phủ đã áp dụng **AES** trong các hệ thống và dịch vụ của họ nhằm đảm bảo an toàn cho thông tin và dữ liệu quan trọng.

Việc sử dụng **AES** không chỉ đem lại sự an tâm cho các tổ chức và cá nhân trong việc bảo vệ thông tin, mà còn thúc đẩy sự phát triển của ngành công nghiệp bảo mật thông tin tại Việt Nam. Các chuyên gia và nhà nghiên cứu ở Việt Nam cũng tham gia vào việc nghiên cứu và phát triển các ứng dụng và giải pháp dựa trên **AES**, góp phần nâng cao năng lực cạnh tranh của ngành công nghiệp công nghệ thông tin Việt Nam trên thị trường toàn cầu.

3. Thuật toán của AES

Thuật toán AES được mô tả khái quát gồm 3 bước như sau:

- 1 Vòng khởi tạo chỉ gồm phép AddRoundKey
- Nr -1 Vòng lặp gồm 4 phép biến đổi lần lượt: SubBytes, ShiftRows, MixColumns, AddRoundKey.

- 1 Vòng cuối gồm các phép biến đổi giống vòng lặp và không có phép MixColumns.

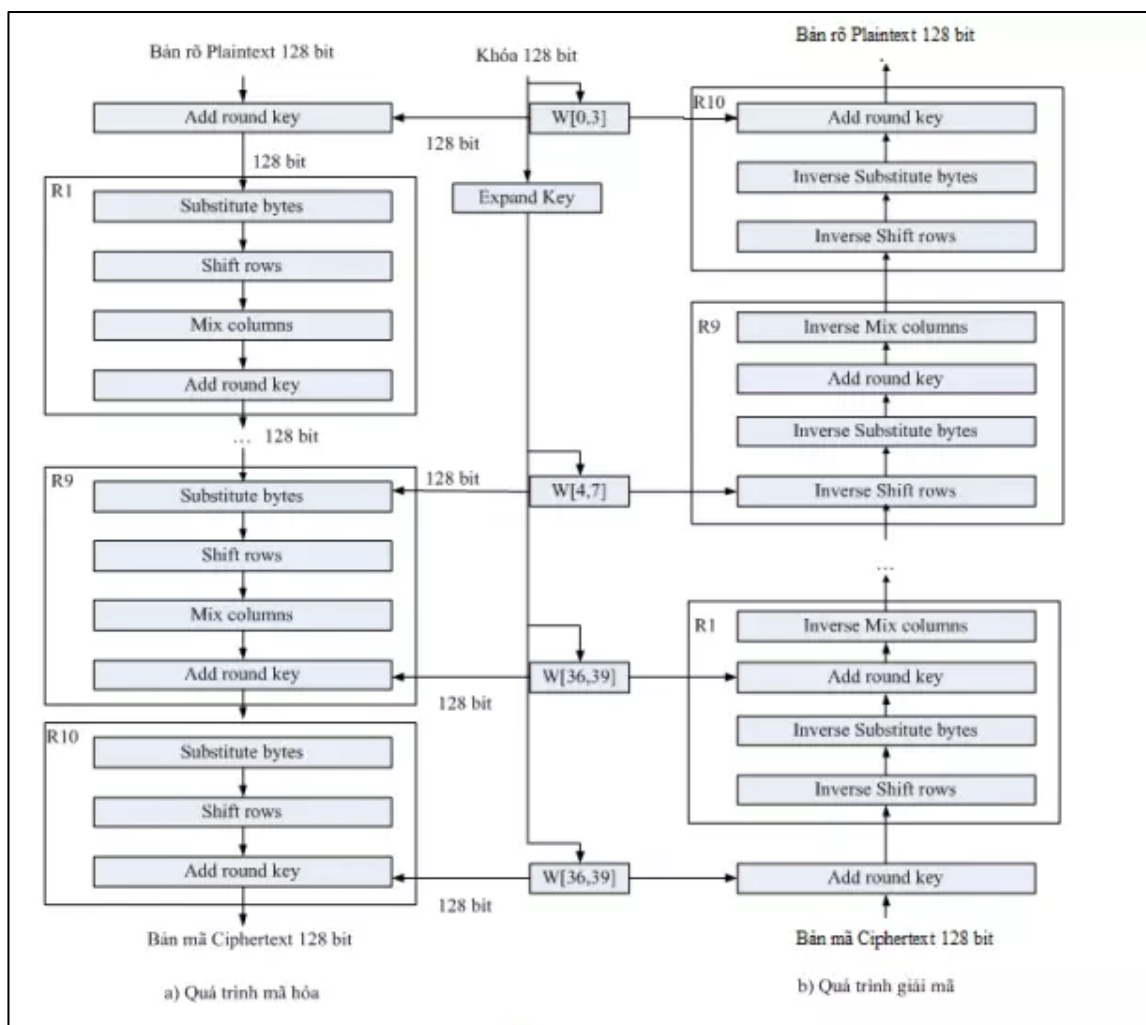
Khái quát: Đầu tiên, các khóa phụ được tạo ra từ khóa chính sử dụng thủ tục sinh khóa Rijndael để sử dụng trong các vòng lặp. Tiếp theo, trong InitialRound, mỗi byte trong state được kết hợp với khóa phụ thông qua phép XOR. Sau đó, trong bước Rounds, các bước SubBytes, ShiftRows và MixColumns được thực hiện để thay thế, đổi chỗ và trộn các cột trong state, kết hợp với khóa phụ. Cuối cùng, trong Final Round, không có bước MixColumns, chỉ có các bước SubBytes, ShiftRows và AddRoundKey được áp dụng.

Thuật toán giải mã AES tương tự như thuật toán mã hóa về cấu trúc, nhưng sử dụng các hàm ngược của quá trình mã hóa. **Có hai chế độ giải mã trong AES:**

- **Cấu trúc giải mã ngược:** Bao gồm vòng khởi tạo, Nr-1 vòng lặp và vòng kết thúc. Trong đó, vòng khởi tạo chỉ sử dụng phép AddRoundKey. Trong vòng lặp, các phép biến đổi ngược lần lượt là InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns. Vòng kết thúc không sử dụng phép InvMixColumns.

- **Cấu trúc giải mã xuôi:** Sắp xếp các phép biến đổi ngược giống hệt với cấu trúc mã hóa. Bao gồm vòng khởi tạo, Nr-1 vòng lặp và vòng kết thúc. Trong đó, vòng khởi tạo sử dụng phép AddRoundKey. Trong vòng lặp, các phép biến đổi ngược lần lượt là InvSubBytes, InvShiftRows, InvMixColumns, AddRoundKey. Vòng kết thúc giống vòng lặp nhưng không sử dụng phép InvMixColumns.

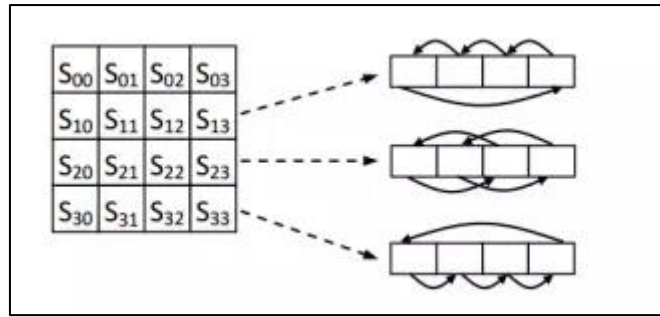
Một điểm khác biệt quan trọng giữa hai cấu trúc giải mã là khóa vòng giải mã. Trong giải mã ngược, khóa vòng giải mã là khóa vòng mã hóa với thứ tự đảo ngược. Trong giải mã xuôi, ngoài việc đảo ngược thứ tự khóa vòng mã hóa, còn phải thực hiện phép InvMixColumns đối với các khóa vòng của vòng lặp giải mã.



Hình 1: Mô hình hoá thuật toán AES

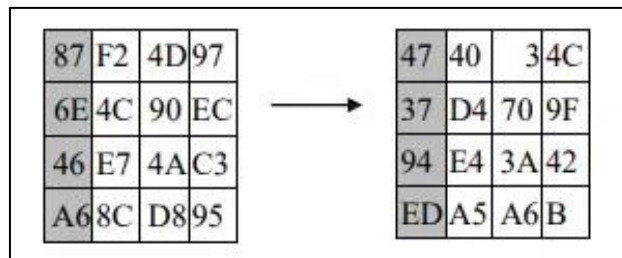
SubBytes(): Phép biến đổi dùng trong phép mã hóa áp dụng lên trạng thái (kết quả mã hóa trung gian, được mô tả dưới dạng một mảng chữ nhật của các byte) sử dụng một bảng thay thế byte phi tuyến (Hộp S – bảng thay thế phi tuyến, được sử dụng trong một số phép thay thế byte và trong quy trình mở rộng khóa, nhằm thực hiện một phép thay thế 1-1 đối với giá trị mỗi byte) trên mỗi byte trạng thái một cách độc lập.

- ShiftRows(): Phép biến đổi dùng trong phép mã hóa áp dụng lên trạng thái bằng cách chuyển dịch vòng ba hàng cuối của trạng thái theo số lượng byte các offset khác nhau.



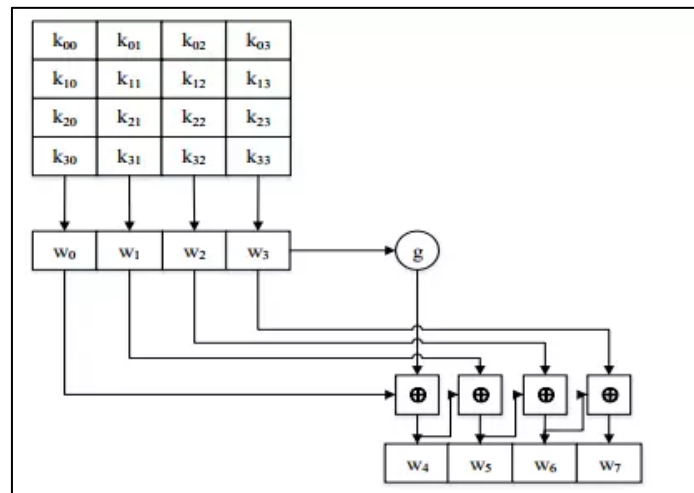
Hình 2: Hàm ShiftRows

- MixColumns(): Phép biến đổi trong phép mã hóa thực hiện bằng cách lấy tất cả các cột trạng thái trộn với dữ liệu của chúng (một cách độc lập nhau) để tạo ra các cột mới.



Hình 3: Hàm MixColumn

- AddRoundKey(): Phép biến đổi trong phép mã hóa và phép giải mã. Trong đó, một khóa vòng (các giá trị sinh ra từ khóa mã bằng quy trình mở rộng khóa) được cộng thêm vào trạng thái bằng phép toán XOR (phép toán hoặc và loại trừ). Độ dài của khóa vòng bằng độ dài của trạng thái.



Hình 4: Mô hình hoá giai đoạn mở rộng khoá

- Mở rộng khóa: Thuật toán AES nhận vào một khóa mã K và thực hiện phép mở rộng khóa để tạo ra một lược đồ khóa. Phép mở rộng khóa tạo ra tổng số $N_b(N_r+1)$ từ. Thuật toán yêu cầu một tập khởi tạo gồm N_b từ và mỗi trong số N_r vòng đòi hỏi N_b từ làm dữ liệu khóa đầu vào. Lược đồ khóa kết quả là một mảng tuyến tính các từ 4 byte.

- Phép giải mã: Các phép biến đổi trong phép mã hóa có thể được đảo ngược và sau đó thực hiện theo chiều ngược lại nhằm tạo ra phép giải mã trực tiếp của thuật toán AES. Các phép biến đổi sử dụng trong phép giải mã gồm: `InvShiftRows()`, `InvSubBytes()`, `InvMixColumns()` và `AddRoundKey()`.

- `InvSubBytes()`: Phép biến đổi `InvSubBytes()` là nghịch đảo của phép thay thế theo byte `SubBytes()`, trong đó sử dụng một hộp-S nghịch đảo áp dụng cho mỗi byte của trạng thái.

- `InvShiftRows()`: Phép biến đổi `InvShiftRows()` là phép biến đổi ngược của `ShiftRows()`. Các byte trong ba từ cuối của trạng thái được dịch vòng theo số byte khác nhau. Ở hàng đầu tiên ($r=0$) không thực hiện phép chuyển dịch, ba hàng dưới cùng được dịch vòng $Nb\text{-}shift(r, Nb)$ byte.

- `InvMixColumns()`: Phép biến đổi `InvMixColumns()` là phép biến đổi ngược của `MixColumns()`. Nó thao tác theo từng cột của trạng thái, xem mỗi cột như một đa thức bốn hạng tử.

- Biến đổi nghịch `AddRoundKey()`: Phép biến đổi `AddRoundKey()` là phép biến đổi thuận nghịch vì nó chỉ áp dụng một phép toán XOR nên nó được thực hiện như nhau ở cả phép mã hóa và phép giải mã.

4. Mô tả thuật toán AES bằng mã nguồn C++

```
unsigned int RotWord(unsigned int w)
{
    unsigned int byte1 = (w >> 24) & 0xFF;
    unsigned int byte234 = w & 0FFFFFF;
    unsigned int rot = (byte234 << 8) | byte1;
    return rot;
}
```

Hàm **RotWord** thực hiện phép dịch vòng trái (rotation) 1 byte trên một từ 32-bit.

- Biến **w** là từ cần thực hiện phép dịch vòng trái.
- **w >> 24** dịch chuyển **w** sang phải 24 bit, để lấy giá trị của byte đầu tiên (byte1).
- **& 0xFF** chỉ lấy 8 bit cuối cùng của byte đó, vì **0xFF** có dạng bin là **11111111**, do đó nó sẽ giữ lại 8 bit cuối cùng và loại bỏ các bit cao hơn.
- **byte234 = w & 0FFFFFF** giữ lại tất cả các byte của từ **w** ngoại trừ byte đầu tiên.
- **rot = (byte234 << 8) | byte1** dịch chuyển byte234 sang trái 8 bit (tương đương với việc dịch vòng trái 1 byte) và sau đó kết hợp nó với byte1 để tạo thành từ kết quả.

```

unsigned int SubWord(unsigned int w)
{
    int S[] = {0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67,
0x2B, 0xFE, 0xD7, 0xAB, 0x76,
                0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4,
0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
                0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5,
0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
                0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12,
0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
                0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B,
0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
                0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB,
0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
                0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9,
0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
                0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6,
0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
                0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7,
0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
                0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE,
0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
                0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3,
0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
                0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56,
0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
                0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD,
0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
                0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35,
0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
                0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E,
0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
                0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99,
0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16};
    unsigned int kq = 0;
    for(int i = 1; i <= 4; i++)
    {
        unsigned int bytei = (w >> 32 - i * 8) & 0xFF;
        unsigned int subB = S[bytei];
        kq = (kq << 8) | subB;
    }
    return kq;
}

```

Hàm **SubWord** thực hiện phép thay thế mỗi byte trong một từ 32-bit bằng giá trị tương ứng từ một bảng tham chiếu được định nghĩa trước.

- Biến **w** là từ 32-bit cần thực hiện phép thay thế.
- Mảng **S** là bảng tham chiếu chứa các giá trị thay thế tương ứng cho các byte từ **00** đến **FF**.
- Vòng lặp **for** chạy qua từng byte của từ **w**.
- Trong mỗi lần lặp, **bytei = (w >> 32 - i * 8) & 0xFF** dịch chuyển và lấy giá trị của byte thứ **i** từ từ **w**.
- **S[bytei]** trả về giá trị thay thế tương ứng cho byte **bytei** từ bảng **S**.
- Kết quả cuối cùng **kq** được cập nhật bằng cách dịch vòng trái 8 bit và kết hợp với giá trị thay thế mới.

```
unsigned int XorRcon(unsigned int w, int j)
{
    int Rc[] = {
        0x8d, 0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b,
0x36, 0x6c, 0xd8, 0xab, 0x4d, 0x9a,
        0x2f, 0x5e, 0xbc, 0x63, 0xc6, 0x97, 0x35, 0x6a, 0xd4, 0xb3,
0x7d, 0xfa, 0xef, 0xc5, 0x91, 0x39
    };
    unsigned int byte1 = (w >> 24) & 0xFF;
    unsigned int kqXor = (byte1 ^ Rc[j]) & 0xFF;
    unsigned int byte234 = w & 0FFFFFF;
    unsigned int kq = (kqXor << 24) | byte234;

    return kq;
}
```

Hàm **XorRcon** thực hiện phép XOR giữa byte đầu tiên của một từ 32-bit và một giá trị từ một mảng tham chiếu được định nghĩa trước. Giá trị tham chiếu được chọn dựa trên chỉ số **j**.

- Biến **w** là từ 32-bit cần thực hiện phép XOR.
- Biến **j** là chỉ số để chọn giá trị từ mảng **Rc**.
- Mảng **Rc** chứa các giá trị được sử dụng để thực hiện phép XOR.
- **byte1** là byte đầu tiên của từ **w**.
- **Rc[j]** là giá trị từ mảng **Rc** được chọn dựa trên chỉ số **j**.
- **kqXor** lưu trữ kết quả của phép XOR giữa **byte1** và **Rc[j]**.
- **byte234** là byte 2, 3 và 4 của từ **w**.
- Kết quả cuối cùng **kq** được tạo ra bằng cách kết hợp **kqXor** và **byte234** lại với nhau.

```

unsigned int G(unsigned int w, int j)
{
    unsigned int rotW = RotWord(w);
    unsigned int subW = SubWord(rotW);
    unsigned int kq = XorRcon(subW, j);
    return kq;
}

```

Hàm **G** thực hiện một loạt các phép biến đổi trên một từ 32-bit **w** dựa trên một chỉ số **j**. Cụ thể:

- **RotWord(w)**: Thực hiện dịch vòng trên từ **w**, nghĩa là dịch các byte sang trái một vị trí. Kết quả của phép dịch này được lưu vào biến **rotW**.
- **SubWord(rotW)**: Thực hiện thay thế mỗi byte trong từ **rotW** bằng một giá trị từ bảng tham chiếu S. Kết quả của phép thay thế này được lưu vào biến **subW**.
- **XorRcon(subW, j)**: Thực hiện phép XOR giữa từ **subW** và một giá trị từ mảng tham chiếu **Rc** dựa trên chỉ số **j**. Kết quả của phép XOR này được lưu vào biến **kq**.

```

unsigned int* KeyExpansion(unsigned int Key[4])
{
    unsigned int *w = new unsigned int[44];
    w[0] = Key[0];
    w[1] = Key[1];
    w[2] = Key[2];
    w[3] = Key[3];
    for(int i = 4; i < 44; i++)
    {
        if(i % 4 == 0) w[i] = G(w[i - 1], i / 4) ^ w[i - 4];
        else w[i] = w[i - 1] ^ w[i - 4];
    }
    return w;
}

```

Hàm **KeyExpansion** thực hiện mở rộng khóa ban đầu **Key** thành một mảng các từ 32-bit **w**. Quá trình mở rộng khóa được thực hiện như sau:

- Các từ đầu tiên của mảng **w** được sao chép từ khóa ban đầu.
- Các từ tiếp theo được tạo ra thông qua việc lặp lại một quy trình, trong đó mỗi từ mới được tính toán dựa trên từ trước đó và từ các vòng trước.
- Trong mỗi vòng lặp, nếu **i** là bội số của 4, tức là **i % 4 == 0**, hàm **G** được áp dụng lên từ trước đó **w[i - 1]** với chỉ số **i / 4**, sau đó được XOR với từ cách đúng 4 từ trước đó **w[i - 4]**. Ngược lại, từ mới được tạo bằng cách XOR từ trước đó **w[i-1]** với từ cách đúng 4 từ trước đó **w[i-4]**.

- Cuối cùng, mảng **w** chứa các từ được mở rộng được trả về.

```
unsigned int* AddRoundKey(unsigned int state[4], unsigned int *K)
{
    unsigned int *kq = new unsigned int[4];
    kq[0] = state [0] ^ K[0];
    kq[1] = state [1] ^ K[1];
    kq[2] = state [2] ^ K[2];
    kq[3] = state [3] ^ K[3];
    return kq;
}
```

Hàm **AddRoundKey** thực hiện phép XOR giữa trạng thái hiện tại và khóa của vòng tròn được cung cấp. Cụ thể:

- Mỗi từ trong trạng thái hiện tại (**state**) được XOR với từ tương ứng trong khóa của vòng tròn (**K**).
- Kết quả được lưu trữ trong một mảng mới **kq**.

```
unsigned int* SubBytes(unsigned int state[4])
{
    unsigned int *kq = new unsigned int[4];
    for (int i = 0; i < 4; i++)
        kq[i] = SubWord(state[i]);
    return kq;
}
```

Hàm **SubBytes** thực hiện thay thế mỗi byte trong trạng thái hiện tại (**state**) bằng giá trị tương ứng từ bảng tham chiếu. Cụ thể:

- Mỗi từ trong trạng thái hiện tại được truyền vào hàm **SubWord** để thay thế từng byte bằng giá trị từ bảng tham chiếu.
- Kết quả được lưu trữ trong một mảng mới **kq**.

```
unsigned int* ShiftRows(unsigned int state[4])
{
    unsigned int *kq = new unsigned int[4];
    for (int i = 0; i < 4; i++) {
        unsigned int byte1 = state[i] & 0xFF000000;
        unsigned int byte2 = state[(i + 1) % 4] & 0xFF0000;
        unsigned int byte3 = state[(i + 2) % 4] & 0xFF00;
        unsigned int byte4 = state[(i + 3) % 4] & 0xFF;
        kq[i] = byte1 | byte2 | byte3 | byte4;
    }
    return kq;
}
```

Hàm **ShiftRows** dịch các hàng của trạng thái hiện tại theo quy tắc cố định. Cụ thể:

- Hàng đầu tiên không được dịch chuyển.
- Hàng thứ hai dịch chuyển sang phải một byte.
- Hàng thứ ba dịch chuyển sang phải hai byte.
- Hàng thứ tư dịch chuyển sang phải ba byte.
- Mỗi byte trong từng hàng được lấy ra và ghép lại theo thứ tự mới để tạo thành trạng thái mới. Cuối cùng, trạng thái mới này được lưu trữ trong một mảng mới **kq** và được trả về.

```
unsigned int Nhan2(unsigned int w)
{
    unsigned int kq = w << 1;
    if(kq > 256) kq = kq ^ 0x11b;
    kq = kq & 0xFF;
    return kq;
}

unsigned int Nhan3(unsigned int w)
{
    unsigned int kq = w ^ Nhan2(w);
    kq = kq & 0xFF;
    return kq;
}
```

Hai hàm **Nhan2** và **Nhan3** thực hiện phép nhân của một số trong trường hữu hạn $GF(2^8)$ với 2 và 3 tương ứng. Trong đó:

- **Nhan2**: Đầu vào là một byte **w**, và nó thực hiện phép nhân của **w** với 2 trong trường hữu hạn $GF(2^8)$. Nếu kết quả vượt quá 255 (0xFF), thì phép XOR với 0x11b được thực hiện để thực hiện modulo của phép nhân trong trường hữu hạn. Cuối cùng, kết quả được giữ lại trong 8 bit thấp nhất và trả về.
- **Nhan3**: Đầu vào là một byte **w**, và nó thực hiện phép nhân của **w** với 3 trong trường hữu hạn $GF(2^8)$. Để thực hiện điều này, nó sử dụng phép XOR của **w** với kết quả của phép nhân 2 (**Nhan2(w)**). Kết quả sau đó được giữ lại trong 8 bit thấp nhất và trả về.

```

unsigned int NhanCot(unsigned int w)
{
    unsigned int kq;
    unsigned int byte1 = (w >> 24) & 0xFF;
    unsigned int byte2 = (w >> 16) & 0xFF;
    unsigned int byte3 = (w >> 8) & 0xFF;
    unsigned int byte4 = w & 0xFF;
    unsigned int kq1 = Nhan2(byte1) ^ Nhan3(byte2) ^ byte3 ^ byte4;
    unsigned int kq2 = byte1 ^ Nhan2(byte2) ^ Nhan3(byte3) ^ byte4;
    unsigned int kq3 = byte1 ^ byte2 ^ Nhan2(byte3) ^ Nhan3(byte4);
    unsigned int kq4 = Nhan3(byte1) ^ byte2 ^ byte3 ^ Nhan2(byte4);

    kq = (kq1 << 24) | (kq2 << 16) | (kq3 << 8) | kq4;
    return kq;
}

```

Hàm **NhanCot** thực hiện phép nhân cột trong thuật toán AES. Đầu vào của hàm là một từ 32-bit, đại diện cho một cột trong trạng thái của khối dữ liệu AES. Cột này được chia thành 4 byte, và mỗi byte sẽ trải qua một loạt các phép nhân và XOR để tạo ra một cột mới.

Để thực hiện điều này:

- Mỗi byte của cột được nhân với các hằng số đa thức đặc biệt theo cơ số 2 trong trường hữu hạn $GF(2^8)$.
- Sau đó, các kết quả của các phép nhân này được kết hợp thông qua phép XOR với các byte khác trong cùng một cột và được sắp xếp lại thành một cột mới.

```

unsigned int* MixColumns(unsigned int state[4])
{
    unsigned int *kq = new unsigned int[4];
    for (int i = 0; i < 4; i++)
        kq[i] = NhanCot(state[i]);
    return kq;
}

void ShowMatrix(unsigned int w[4])
{
    for (int i = 0; i < 4; i++)
    {
        printf("\n\t"); ShowWord(w[i]);
    }
}

```

Hàm **MixColumns** thực hiện pha MixColumns trong thuật toán AES. Đầu vào của hàm là một ma trận 4x4, trong đó mỗi cột đại diện cho một từ 32-bit, biểu diễn một cột trong trạng thái của khối dữ liệu AES. Mỗi cột này sẽ được xử lý bằng cách thực hiện phép nhân cột, như đã mô tả trong hàm **NhanCot**. Kết quả sau cùng là một ma trận mới, trong đó mỗi cột đã được xử lý thông qua pha MixColumns.

Hàm **MahoaAES** thực hiện quá trình mã hóa một khối dữ liệu AES sử dụng một khóa đã cho.

- **KeyExpansion**: Đầu tiên, khóa được mở rộng thành một tập hợp các từ khóa con sử dụng hàm **KeyExpansion**.
- **AddRoundKey (Vòng 0)**: Khóa mở rộng đầu tiên được XOR với trạng thái ban đầu.
- **Vòng lặp 9 vòng**: Trong mỗi vòng, trạng thái được thực hiện các bước SubBytes, ShiftRows, MixColumns và AddRoundKey. Cụ thể:
 - **SubBytes**: Mỗi byte trong trạng thái được thay thế bằng giá trị tương ứng trong hộp S-box.
 - **ShiftRows**: Các byte trong mỗi hàng của trạng thái được dịch vòng sang phải.
 - **MixColumns**: Các cột trong trạng thái được kết hợp theo luật của pha MixColumns.
 - **AddRoundKey**: Trạng thái được XOR với từ khóa con tương ứng từ tập mở rộng khóa.
- **Vòng cuối (Vòng 10)**: Trong vòng cuối cùng, không có bước MixColumns.
- Trạng thái cuối cùng sau vòng lặp được trả về.

```
unsigned int* InvShiftRows(unsigned int state[4])
{
    unsigned int *kq = new unsigned int[4];
    for (int i = 0; i < 4; i++)
    {
        unsigned int byte1 = state[i] & 0xFF000000;
        unsigned int byte2 = state[(i + 3) % 4] & 0xFF0000;
        unsigned int byte3 = state[(i + 2) % 4] & 0xFF00;
        unsigned int byte4 = state[(i + 1) % 4] & 0xFF;

        kq[i] = byte1 | byte2 | byte3 | byte4;
    }
    return kq;
}
```

Hàm **InvShiftRows** thực hiện quá trình dịch các hàng của trạng thái ngược trở lại so với hàm **ShiftRows** trong quá trình giải mã AES.

Trong quá trình mã hóa, các byte trong mỗi hàng của trạng thái được dịch vòng sang phải. Trong quá trình giải mã, chúng ta cần dịch các hàng ngược lại, nghĩa là:

- Hàng 1 không thay đổi.
- Hàng 2 dịch vòng sang phải 1 byte.
- Hàng 3 dịch vòng sang phải 2 byte.
- Hàng 4 dịch vòng sang phải 3 byte.

Dưới đây là cách thực hiện trong hàm **InvShiftRows**:

- Hàng 1 không thay đổi.
- Hàng 2 lấy byte thứ 4 của hàng trước đó.
- Hàng 3 lấy byte thứ 3 của hàng trước đó.
- Hàng 4 lấy byte thứ 2 của hàng trước đó.

```
unsigned int InvSubWord(unsigned int w)
{
    int InvS[] = {
        0x52, 0x09, 0x6a, 0xd5, 0x30, 0x36, 0xa5, 0x38, 0xbf, 0x40, 0xa3,
        0x9e, 0x81, 0xf3, 0xd7, 0xfb,
        0x7c, 0xe3, 0x39, 0x82, 0x9b, 0x2f, 0xff, 0x87, 0x34, 0x8e, 0x43,
        0x44, 0xc4, 0xde, 0xe9, 0xcb,
        0x54, 0x7b, 0x94, 0x32, 0xa6, 0xc2, 0x23, 0x3d, 0xee, 0x4c, 0x95,
        0x0b, 0x42, 0xfa, 0xc3, 0x4e,
        0x08, 0x2e, 0xa1, 0x66, 0x28, 0xd9, 0x24, 0xb2, 0x76, 0x5b, 0xa2,
        0x49, 0x6d, 0x8b, 0xd1, 0x25,
        0x72, 0xf8, 0xf6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xd4, 0xa4, 0x5c,
        0xcc, 0x5d, 0x65, 0xb6, 0x92,
        0x6c, 0x70, 0x48, 0x50, 0xfd, 0xed, 0xb9, 0xda, 0x5e, 0x15, 0x46,
        0x57, 0xa7, 0x8d, 0x9d, 0x84,
        0x90, 0xd8, 0xab, 0x00, 0x8c, 0xbc, 0xd3, 0x0a, 0xf7, 0xe4, 0x58,
        0x05, 0xb8, 0xb3, 0x45, 0x06,
        0xd0, 0x2c, 0x1e, 0x8f, 0xca, 0x3f, 0x0f, 0x02, 0xc1, 0xaf, 0xbd,
        0x03, 0x01, 0x13, 0x8a, 0x6b,
        0x3a, 0x91, 0x11, 0x41, 0x4f, 0x67, 0xdc, 0xea, 0x97, 0xf2, 0xcf,
        0xce, 0xf0, 0xb4, 0xe6, 0x73,
        0x96, 0xac, 0x74, 0x22, 0xe7, 0xad, 0x35, 0x85, 0xe2, 0xf9, 0x37,
        0xe8, 0x1c, 0x75, 0xdf, 0x6e,
        0x47, 0xf1, 0x1a, 0x71, 0x1d, 0x29, 0xc5, 0x89, 0x6f, 0xb7, 0x62,
        0x0e, 0xaa, 0x18, 0xbe, 0x1b,
        0xfc, 0x56, 0x3e, 0x4b, 0xc6, 0xd2, 0x79, 0x20, 0x9a, 0xdb, 0xc0,
        0xfe, 0x78, 0xcd, 0x5a, 0xf4,
```

```

        0x1f, 0xdd, 0xa8, 0x33, 0x88, 0x07, 0xc7, 0x31, 0xb1, 0x12, 0x10,
0x59, 0x27, 0x80, 0xec, 0x5f,
        0x60, 0x51, 0x7f, 0xa9, 0x19, 0xb5, 0x4a, 0x0d, 0x2d, 0xe5, 0x7a,
0x9f, 0x93, 0xc9, 0x9c, 0xef,
        0xa0, 0xe0, 0x3b, 0x4d, 0xae, 0x2a, 0xf5, 0xb0, 0xc8, 0xeb, 0xbb,
0x3c, 0x83, 0x53, 0x99, 0x61,
        0x17, 0x2b, 0x04, 0x7e, 0xba, 0x77, 0xd6, 0x26, 0xe1, 0x69, 0x14,
0x63, 0x55, 0x21, 0x0c, 0x7d
    };
    unsigned int kq = 0;
    for(int i = 1; i <= 4; i++)
    {
        unsigned int bytei = (w >> (32 - i * 8)) & 0xFF;
        unsigned int subB = InvS[bytei];

        kq = (kq << 8) | subB;
    }
    return kq;
}

```

Hàm **InvSubWord** thực hiện thay thế mỗi byte trong từ (word) đầu vào bằng giá trị tương ứng từ bảng tham chiếu ngược của AES.

Cụ thể, nó lấy từ 32-bit (word) đầu vào và thay thế từng byte trong từ đó bằng giá trị tương ứng trong bảng tham chiếu ngược. Bảng tham chiếu này được sử dụng trong quá trình giải mã AES để thực hiện ngược lại việc thay thế byte trong quá trình mã hóa.

```

unsigned int* InvSubBytes(unsigned int state[4])
{
    unsigned int *kq = new unsigned int[4];
    for (int i = 0; i < 4; i++)
        kq[i] = InvSubWord(state[i]);

    return kq;
}

```

Hàm **InvSubBytes** thực hiện việc thay thế từng byte trong mỗi từ 32-bit của trạng thái đầu vào bằng giá trị tương ứng từ bảng tham chiếu ngược của AES, sử dụng hàm **InvSubWord**.

```

unsigned int Nhan9(unsigned int w)
{
    unsigned int kq = (w << 3) ^ w;
    if(kq > (256 << 2)) kq = kq ^ (0x11b << 2);
    if(kq > (256 << 1)) kq = kq ^ (0x11b << 1);
}

```

```

    if(kq > 256) kq = kq ^ 0x11b;
    kq = kq & 0xFF;
    return kq;
}

unsigned int NhanB(unsigned int w)
{
    unsigned int kq = (w << 3) ^ (w << 1) ^ w;
    if(kq > (256 << 2)) kq = kq ^ (0x11b << 2);
    if(kq > (256 << 1)) kq = kq ^ (0x11b << 1);
    if(kq > 256) kq = kq ^ 0x11b;

    kq = kq & 0xFF;
    return kq;
}

unsigned int NhanD(unsigned int w)
{
    unsigned int kq = (w << 3) ^ (w << 2) ^ w;
    if(kq > (256 << 2)) kq = kq ^ (0x11b << 2);
    if(kq > (256 << 1)) kq = kq ^ (0x11b << 1);
    if(kq > 256) kq = kq ^ 0x11b;
    kq = kq & 0xFF;
    return kq;
}

unsigned int NhanE(unsigned int w)
{
    unsigned int kq = (w << 3) ^ (w << 2) ^ (w << 1);
    if(kq > (256 << 2)) kq = kq ^ (0x11b << 2);
    if(kq > (256 << 1)) kq = kq ^ (0x11b << 1);
    if(kq > 256) kq = kq ^ 0x11b;
    kq = kq & 0xFF;
    return kq;
}

```

Các hàm **Nhan9**, **NhanB**, **NhanD**, và **NhanE** thực hiện phép nhân trong trường hữu hạn $GF(2^8)$ theo các hệ số cụ thể như sau:

- Hàm **Nhan9**: Nhân với hằng số 9 trong trường hữu hạn $GF(2^8)$.
- Hàm **NhanB**: Nhân với hằng số B (1011) trong trường hữu hạn $GF(2^8)$.
- Hàm **NhanD**: Nhân với hằng số D (1101) trong trường hữu hạn $GF(2^8)$.
- Hàm **NhanE**: Nhân với hằng số E (1110) trong trường hữu hạn $GF(2^8)$.

Các hằng số này được chọn để thực hiện phép nhân trong trường hữu hạn $GF(2^8)$ theo đúng qui tắc của AES. Cụ thể, mỗi hàm sẽ thực hiện các phép dịch trái và phép XOR để tính toán kết quả. Đồng thời, sau mỗi lần dịch, nếu giá trị vượt quá 256 thì sẽ thực hiện phép XOR với 0x11b để đảm bảo kết quả thuộc trong trường hữu hạn $GF(2^8)$. Cuối cùng, kết quả được giữ lại trong khoảng từ 0 đến 255 bằng cách AND với 0xFF.

```
unsigned int InvNhanCot(unsigned int w)
{
    unsigned int kq;
    unsigned int byte1 = (w >> 24) & 0xFF;
    unsigned int byte2 = (w >> 16) & 0xFF;
    unsigned int byte3 = (w >> 8) & 0xFF;
    unsigned int byte4 = w & 0xFF;
    unsigned int kq1 = NhanE(byte1) ^ NhanB(byte2) ^ NhanD(byte3) ^ Nhan9(byte4);
    unsigned int kq2 = Nhan9(byte1) ^ NhanE(byte2) ^ NhanB(byte3) ^ NhanD(byte4);
    unsigned int kq3 = NhanD(byte1) ^ Nhan9(byte2) ^ NhanE(byte3) ^ NhanB(byte4);
    unsigned int kq4 = NhanB(byte1) ^ NhanD(byte2) ^ Nhan9(byte3) ^ NhanE(byte4);

    kq = (kq1 << 24) | (kq2 << 16) | (kq3 << 8) | kq4;
    return kq;
}
```

Hàm **InvNhanCot** thực hiện phép nhân ngược của phép trộn cột trong quá trình giải mã AES. Để thực hiện phép nhân ngược này, nó sử dụng các hàm nhân ngược như **Nhan9**, **NhanB**, **NhanD**, và **NhanE** để tính toán ngược lại các cột.

```
unsigned int* InvMixColumns(unsigned int state[4])
{
    unsigned int *kq = new unsigned int[4];
    for (int i = 0; i < 4; i++)
        kq[i] = InvNhanCot(state[i]);
    return kq;
}
```

Hàm **InvMixColumns** thực hiện phép trộn cột ngược trong quá trình giải mã AES. Đối với mỗi cột trong trạng thái, nó sử dụng hàm **InvNhanCot** để tính toán giá trị ngược của cột đó. Kết quả cuối cùng là trạng thái mới sau khi đã thực hiện phép trộn cột ngược trên tất cả các cột.

```
unsigned int* GiaimaAES(unsigned int C[4], unsigned int key[4])
{
    unsigned int *w = KeyExpansion(key);
    unsigned int *state = AddRoundKey(C, &w[40]);
    for (int j = 1; j <= 9; j++)
    {
```



```

        state = InvShiftRows(state);
        state = InvSubBytes(state);
        state = AddRoundKey(state, &w[40 - 4*j]);
        state = InvMixColumns(state);
    }
    state = InvShiftRows(state);
    state = InvSubBytes(state);
    state = AddRoundKey(state, &w[0]);
    unsigned int *kq = new unsigned int[4];
    kq = state;
    return kq;
}

```

Hàm **GiaimaAES** thực hiện quá trình giải mã AES. Đầu vào là **C**, trạng thái đã được mã hóa cần giải mã, và **key**, khóa sử dụng để giải mã.

Quá trình giải mã bao gồm các bước ngược lại so với quá trình mã hóa:

- **AddRoundKey cuối cùng:** Đầu tiên, khóa round cuối cùng được thêm vào trạng thái.
- **Lặp qua các vòng ngược lại từ 9 đến 1:**
- **InvMixColumns:** Trạng thái được truyền qua hàm InvMixColumns để thực hiện phép trộn cột ngược.
- **AddRoundKey:** Sau đó, khóa round tương ứng được thêm vào trạng thái.
- **InvSubBytes:** Tiếp theo, trạng thái được truyền qua hàm InvSubBytes để thực hiện phép thay thế ngược của byte.
- **InvShiftRows:** Cuối cùng, trạng thái được truyền qua hàm InvShiftRows để thực hiện phép dịch hàng ngược.
- **Round đầu tiên (vòng 0):**
- **AddRoundKey:** Trạng thái được thêm vào khóa round đầu tiên.
- **InvSubBytes:** Tiếp theo, trạng thái được truyền qua hàm InvSubBytes để thực hiện phép thay thế ngược của byte.
- **InvShiftRows:** Cuối cùng, trạng thái được truyền qua hàm InvShiftRows để thực hiện phép dịch hàng ngược.

```

void ShowRoundKeys(unsigned int* roundKeys) {
    printf("\nRound Keys:\n");
    for(int i = 0; i < 11; i++) {
        printf("Round %d:\n", i);
        for(int j = 0; j < 4; j++) {
            printf("\t");
            ShowWord(roundKeys[i * 4 + j]);
            printf("\n");
        }
    }
}

```

Hàm **ShowRoundKeys** được sử dụng để hiển thị các khóa round trong quá trình mã hóa và giải mã AES. Đầu vào của hàm là một con trỏ tới mảng chứa các khóa round.

- Hàm này sẽ lặp qua từng khóa round và hiển thị chúng dưới dạng hexa. Cụ thể:
- Vòng lặp bên ngoài (i) lặp qua mỗi round, từ 0 đến 10.
- Bên trong vòng lặp bên ngoài, có một vòng lặp khác (j) để lặp qua từng phần tử trong mỗi khóa round.
- Mỗi phần tử được hiển thị dưới dạng hexa bằng cách gọi hàm **ShowWord** và sau đó xuống dòng.
- Kết quả sẽ hiển thị mỗi khóa round trên một dòng mới và mỗi byte trong khóa round sẽ được hiển thị dưới dạng hexa.

CHƯƠNG II: GIỚI THIỆU VỀ OPENSSL



Hình 5: Logo OpenSSL

1. Giới thiệu:

OpenSSL là một thư viện phần mềm cho các ứng dụng bảo mật truyền thông qua mạng máy tính chống nghe lén hoặc cần phải xác định phe truyền thông ở bên đầu kia. **OpenSSL** được sử dụng rộng rãi trong các máy chủ web Internet, trong phát triển phần mềm, phục vụ phần lớn tất cả các ứng dụng nói chung.

2. Lịch sử:

Dự án **OpenSSL** được thành lập vào năm 1998 để cung cấp một bộ công cụ mã hóa miễn phí cho các đoạn code chạy trên Internet. Nó dựa trên một phiên bản của **SSLeay** được tạo ra bởi **Eric Andrew Young** và **Tim Hudson**, kết thúc việc phát triển vào ngày 17 tháng 12 năm 1998, khi cả hai đều đi làm việc cho **RSA Security**. Các thành viên sáng lập ban đầu là **Mark Cox**, **Ralf Engelschall**, **Stephen Henson**, **Ben Laurie** và **Paul Sutton**.

Năm 2018, phiên bản **OpenSSL** nhảy từ 1.1.1 đến 3.0.0, bỏ qua số 2 để tránh xung đột với một trong số các mô-đun của **OpenSSL**. Phiên bản 3.0.0 là phiên bản đầu tiên sử dụng **Giấy phép Apache**¹.

Tính đến tháng 5 năm 2019, ủy ban quản lý **OpenSSL** bao gồm bảy người và có mười bảy nhà phát triển có quyền truy cập vào mã nguồn (nhiều trong số đó cũng là thành viên của ủy ban quản lý **OpenSSL**). Chỉ có hai nhân viên toàn thời gian và phần còn lại là tình nguyện viên.

3. Ưu và nhược điểm:

3.1. Ưu điểm:

Bảo mật:

Mục tiêu chính của chứng chỉ **SSL** là mã hóa thông tin để chỉ những người nhận dự kiến mới có thể đọc được. Thông tin được truyền qua Internet có khả năng cao lọt vào tay của các bên thứ ba. Nhờ việc mã hoá bằng chứng chỉ **SSL**, các ký tự ngẫu nhiên

được chèn vào chúng. Ngay cả khi những kẻ xâm nhập có thể có được những thông tin này, họ sẽ không thể hiểu được chúng. Do đó, **SSL** rất lý tưởng trong việc bảo vệ các thông tin nhạy cảm như ID người dùng, mật khẩu và số thẻ tín dụng.

Xác thực:

Như đã đề cập trước đó, dữ liệu truyền qua nhiều bên trong internet. Do đó, nó có thể bị các bên thứ ba truy cập ngoài ý muốn. **SSL** đảm bảo rằng bất kỳ thông tin nào có trên trang web của bạn đều đến được đúng máy chủ. Chính vì thế, cần một **Trusted Third Party**², trong trường hợp này là **SSL Certificate authority**³, để đảm bảo rằng nhà cung cấp chứng chỉ **SSL** đã được tin cậy.

Độ tin cậy:

Một trang web xác thực **SSL** sẽ hiển thị một ổ khóa màu xanh lục trên thanh địa chỉ. Khóa này đề cập rằng trang web đã thực hiện các biện pháp bảo mật và đủ tin cậy để thực hiện các giao dịch.

Ngăn chặn lừa đảo:

Đôi khi, người dùng có thể nhận được email lừa đảo (Thường dưới dạng quảng cáo và xác nhận gửi hàng) đưa các liên kết đến một trang web khác. Mục đích duy nhất của các trang web này là thu thập thông tin nhạy cảm như chi tiết thẻ tín dụng. Tuy nhiên, các trang web này gần như không thể có được chứng chỉ **SSL** xác thực.

Thanh toán trực tuyến:

Tất cả các ngành công nghiệp thẻ thanh toán đều yêu cầu các trang web phải có **chứng chỉ SSL** với mã hóa tối thiểu 128bit để chấp nhận thanh toán. Nếu không có **chứng chỉ SSL** thích hợp, các trang web sẽ không thể chấp nhận thanh toán từ thẻ tín dụng.

Yêu cầu phần mềm:

SSL không đòi hỏi máy khách phải cài đặt phần mềm riêng. Điều duy nhất cần là có kết nối Internet thông qua trình duyệt web tiêu chuẩn. Do đó, không cần phải mua, bảo trì hoặc quản lý phần mềm riêng biệt. Điều này mang lại lợi ích về chi phí cho cả các tổ chức nhỏ và lớn.

3.2. Nhược điểm:

Hiệu suất:

Quá trình mã hóa và giải mã dữ liệu khi sử dụng **SSL** đòi hỏi thêm tài nguyên máy tính và thời gian xử lý. Mỗi lần một trang web yêu cầu trình duyệt của bạn tải một trang đã được bảo vệ bằng **SSL**, dữ liệu cần được mã hóa trước khi được gửi đi và sau đó giải mã khi nhận được. Điều này có thể làm tăng thời gian tải trang, đặc biệt là trên các trang web có nhiều hình ảnh hoặc nội dung động.

Tuy nhiên, ảnh hưởng của việc sử dụng **SSL** đối với tốc độ trang web thường không quá đáng kể đối với người dùng cá nhân hoặc các trang web có lượng truy cập nhỏ. Chỉ khi có một lượng lớn người truy cập cùng lúc, như trong trường hợp của các trang web thương mại lớn, sự chậm trễ này mới trở nên đáng kể.

Chi phí:

Việc mua và thiết lập **chứng chỉ SSL** có thể tốn kém. Điều này chủ yếu xuất phát từ các chi phí liên quan đến bảo trì và quá trình xác minh. Tuy nhiên, một số công ty lưu trữ cung cấp chứng chỉ SSL miễn phí. Tuy nhiên, việc sử dụng chứng chỉ SSL miễn phí thường không được khuyến khích vì nhiều lý do khác nhau.

Chi phí của một **chứng chỉ SSL** có thể thay đổi tùy thuộc vào nhiều yếu tố, bao gồm mức độ xác minh danh tính và số lượng miền và miền phụ mà chứng chỉ bao gồm.

Một số loại **chứng chỉ SSL** có thể bao gồm:

- **Single Domain SSL Certificate:** Chỉ bảo vệ một domain duy nhất.
- **Wildcard SSL Certificate:** Bảo vệ một domain và tất cả các subdomain của nó.
- **Multi-Domain SSL Certificate (SAN Certificate):** Bảo vệ nhiều domain hoặc subdomain khác nhau.

Gia hạn:

Chứng chỉ SSL có thời hạn, điều này có nghĩa là chứng chỉ có hiệu lực trong một khoảng thời gian nhất định. Khi thời hạn của chứng chỉ hết hạn, trình duyệt web sẽ không còn tin tưởng vào tính bảo mật của trang web nữa, và có thể hiển thị cảnh báo cho người dùng.

Bộ nhớ đệm:

Khi nội dung được mã hóa, có thể xảy ra vấn đề với bộ nhớ cache, đặc biệt là khi hệ thống bộ nhớ cache proxy được thiết lập phức tạp trên trình duyệt web. Để giải quyết vấn đề này, cần có một máy chủ được thêm vào để giải mã dữ liệu trước khi nó đến máy chủ bộ nhớ cache. Điều này đảm bảo rằng tất cả dữ liệu được mã hóa đều được giải mã đúng trước khi người dùng truy cập vào trang web, đảm bảo tính bảo mật của dữ liệu.

Các biến chứng của giao thức:

Nếu triển khai **chứng chỉ SSL** không đúng cách, các tệp sẽ được truyền qua **HTTPS** sẽ thay đổi sang **HTTP**. Điều này sẽ gây ra một cảnh báo cho người dùng, thông báo rằng dữ liệu của họ không được bảo vệ.

Hỗ trợ ứng dụng:

Ban đầu, **SSL** chỉ hỗ trợ các ứng dụng web. Bất kỳ ứng dụng nào khác đều cần mua mô-đun từ các nhà cung cấp ứng dụng. Hơn nữa, quá trình thiết lập không dễ dàng và yêu cầu sự thay đổi trong phần mềm nội bộ.

CHƯƠNG III: XÂY DỰNG, CÀI ĐẶT VÀ LÀM VIỆC VỚI OPENSSL

1. Xây dựng và cài đặt OpenSSL cho Windows 10 x64

Microsoft Windows (Windows) là một họ hệ điều hành dựa trên giao diện người dùng đồ họa được phát triển và được phân phối bởi Microsoft.

Trong đó, **Windows 10** là hệ điều hành phổ biến nhất trên thị trường, với một cộng đồng người dùng lớn gồm cả người dùng cơ bản, người dùng doanh nghiệp và nhà phát triển. Điều này tạo ra một thị trường phát triển lớn cho **Windows**, các nhà phát triển luôn hướng tới thị trường này.

Windows 10 x64 là phiên bản của hệ điều hành **Windows** được thiết kế để chạy trên môi trường 64-bit. Điều này có nghĩa là **Windows 10 x64** hỗ trợ cả các ứng dụng và hệ thống có kiến trúc 64-bit, cho phép tận dụng hiệu suất và khả năng xử lý tốt hơn so với phiên bản 32-bit.

Chúng mình lựa chọn **Windows 10 x64** như là một điều hiển nhiên vì sự phổ biến của nó, tuy nhiên, không phủ nhận sự tồn tại của **Windows 10 x32** hay **Windows 11**. Và cụ thể, trong bài báo cáo này chúng mình sẽ trình bày về **OpenSSL** trên **Windows 10 x64**.

Công cụ phát triển mạnh mẽ:

Với bề dày lịch sử và sự đảm bảo về hỗ trợ trong thời gian dài, **Windows** chứng minh mình là môi trường đáng tin cậy cho các nhà phát triển. Chính vì thế, Windows trở thành môi trường phát triển gần như là tiêu chuẩn cho mọi nhà phát triển.

Bản thân **Microsoft** cũng cung cấp một loạt các công cụ phát triển mạnh mẽ như **Visual Studio** và **Visual Studio Code**.

Hỗ trợ đa nền tảng:

Windows 10 hỗ trợ phát triển các ứng dụng đa nền tảng thông qua các công nghệ như .NET Framework, UWP (Universal Windows Platform), và Win32. Điều này cho phép nhà phát triển tạo ra các ứng dụng có thể chạy trên nhiều thiết bị và hệ điều hành khác nhau.

Phức tạp trong quá trình cài đặt và cấu hình:

Windows 10 có thể đòi hỏi một quá trình cài đặt và cấu hình phức tạp, đặc biệt là khi làm việc với các môi trường phát triển phức tạp hoặc các dự án lớn.

Bảo mật và quản lý:

Windows 10 tồn tại một số vấn đề về bảo mật và quản lý, đặc biệt là khi làm việc với các ứng dụng doanh nghiệp hoặc trong môi trường có yêu cầu bảo mật nghiêm ngặt.

Tóm lại: Windows vẫn là môi trường hướng tới đa phần người dùng, đa dụng về chức năng, nên không thể đòi hỏi tối ưu một cách tối đa cho một công việc cụ thể.

1.1 Tải về mã nguồn của OpenSSL

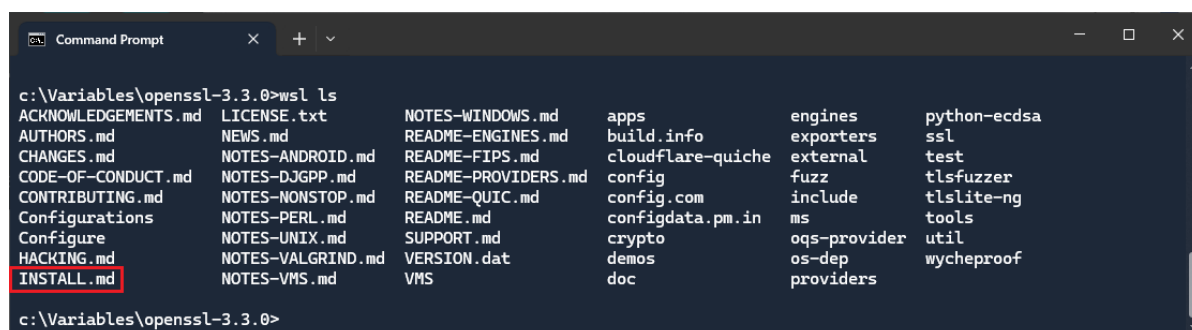
Trước tiên, ta cần tải về mã nguồn của **OpenSSL** từ trang web chính thức hoặc từ kho lưu trữ mã nguồn phổ biến như **GitHub**:

- [\[Downloads \] - /source/index.html \(openssl.org\)](https://www.openssl.org/source/index.html) [tr.63]
- [Releases · openssl/openssl \(github.com\)](https://github.com/openssl/openssl) [tr.63]

1.2. Giải nén

Sau khi tải về, ta giải nén tập tin mã nguồn và di chuyển vào thư mục chứa mã nguồn vừa giải nén.

Chúng ta sẽ quan tâm đến file **INSTALL.md**



Hình 6: Giải nén mã nguồn OpenSSL

Ở đây, mình đã giải nén mã nguồn của **OpenSSL** trong folder:

C:\Variables\openssl-3.3.0

File **INSTALL.md** chứa đầy đủ thông tin hướng dẫn về việc xây dựng và cài đặt **OpenSSL** trên các nền tảng khác nhau, trong bài báo cáo này cũng là làm theo những hướng dẫn này để thực hiện.

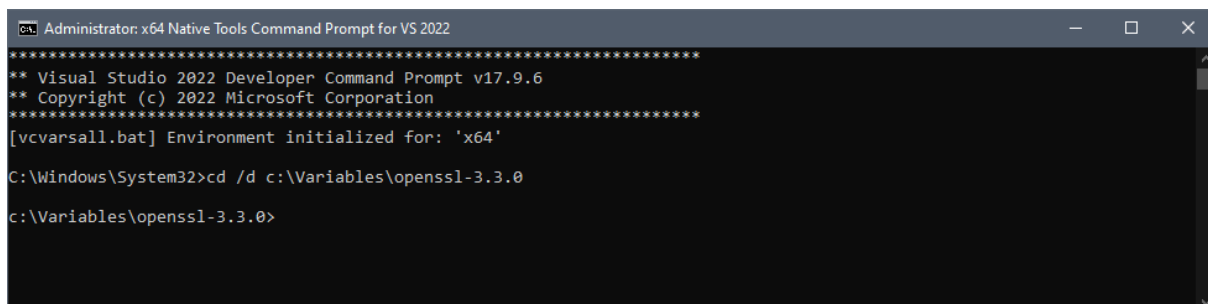
1.3. Sử dụng Perl để configure trong Visual Studio Command Prompt

Sau khi đã [cài đặt và xác định Perl đã tồn tại trong biến môi trường](#)^[tr.46]:

Mở [x64 Native Tools Command Prompt for VS 2022](#)^[tr.51]

(Ở đây nhất thiết phải sử dụng **x64 Native Tools** và dùng quyền quản trị viên, bởi vì chúng ta đang xây dựng chương trình cho **Windows x64**)

Sau khi đã khởi động **x64 Native Tools** cùng quyền **Administrator**, thực hiện di chuyển đến folder chứa mã nguồn của **OpenSSL**



```
Administrator: x64 Native Tools Command Prompt for VS 2022
*****
** Visual Studio 2022 Developer Command Prompt v17.9.6
** Copyright (c) 2022 Microsoft Corporation
*****
[vcvarsall.bat] Environment initialized for: 'x64'
C:\Windows\System32>cd /d c:\Variables\openssl-3.3.0
c:\Variables\openssl-3.3.0>
```

Hình 7: *x64 Native Tools Command Prompt for VS 2022*

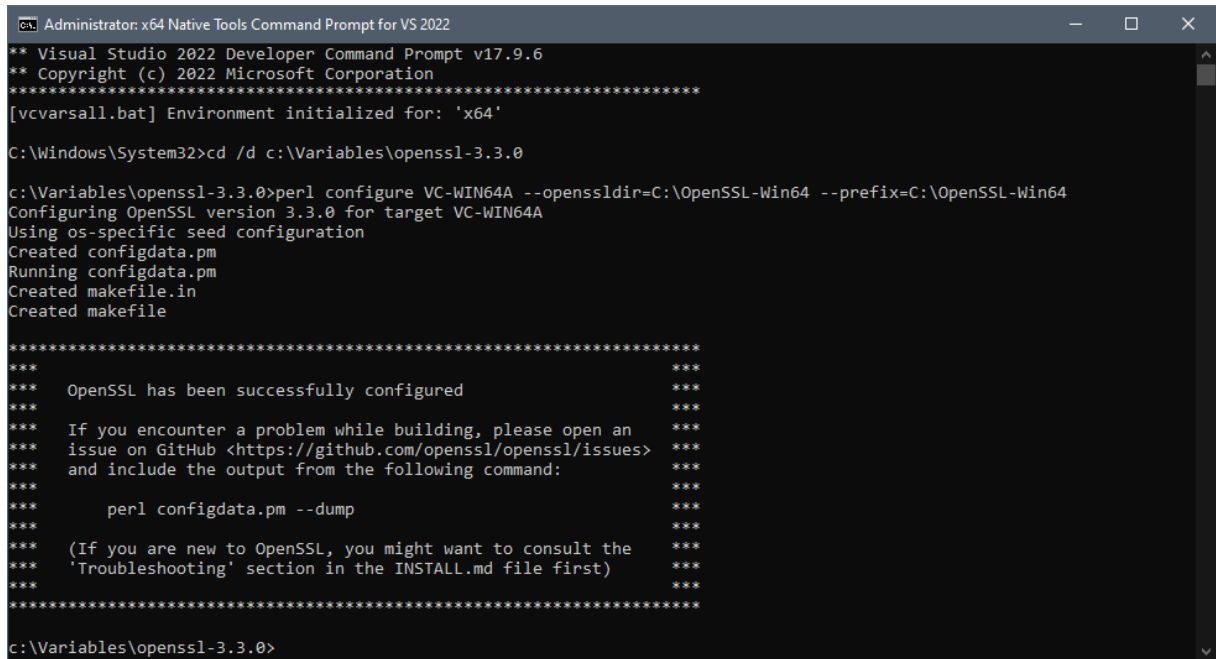
Dùng câu lệnh **Perl** để tạo file makefile:

```
perl configure VC-WIN64A --openssldir=C:\OpenSSL-Win64 --prefix=C:\OpenSSL-Win64
```

Câu lệnh này xác định 3 tham số:

- Openssldir: thư mục cài đặt của **OpenSSL**
- Prefix: thư mục lưu các header và thư viện đi cùng của **OpenSSL**

- **VC-WIN64A** là đối tượng hệ điều hành, tương đương Windows 64-bit.



```
Administrator: x64 Native Tools Command Prompt for VS 2022
** Visual Studio 2022 Developer Command Prompt v17.9.6
** Copyright (c) 2022 Microsoft Corporation
*****
[vcvarsall.bat] Environment initialized for: 'x64'
C:\Windows\System32>cd /d c:\Variables\openssl-3.3.0
c:\Variables\openssl-3.3.0>perl configure VC-WIN64A --openssl-dir=C:\OpenSSL-Win64 --prefix=C:\OpenSSL-Win64
Configuring OpenSSL version 3.3.0 for target VC-WIN64A
Using os-specific seed configuration
Created configdata.pm
Running configdata.pm
Created makefile.in
Created makefile

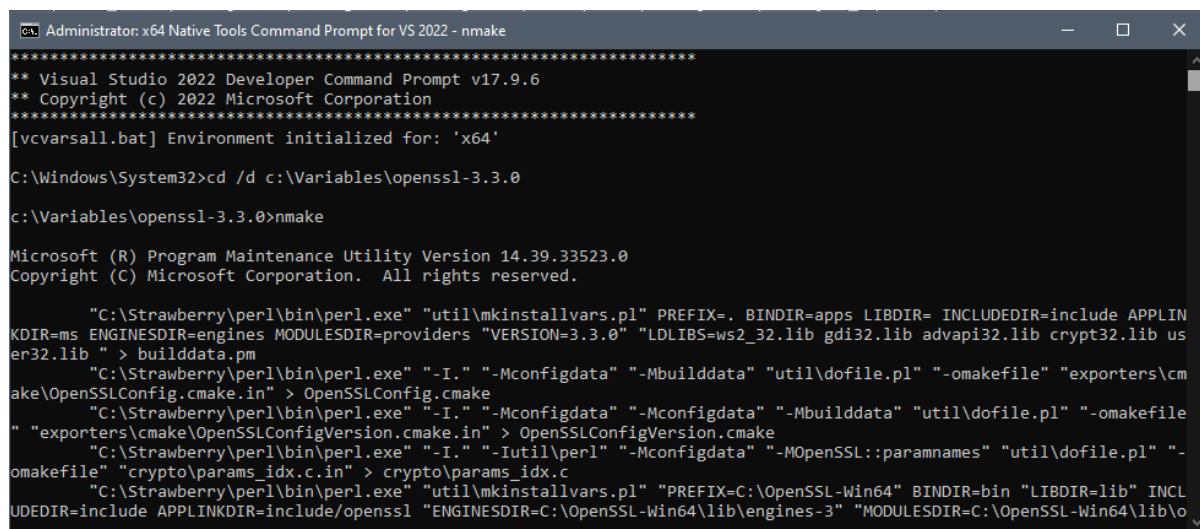
*****
***                                     ***
***  OpenSSL has been successfully configured  ***
***                                     ***
***  If you encounter a problem while building, please open an  ***
***  issue on GitHub <https://github.com/openssl/openssl/issues> ***
***  and include the output from the following command:  ***
***                                     ***
***      perl configdata.pm --dump  ***
***                                     ***
***  (If you are new to OpenSSL, you might want to consult the  ***
***  'Troubleshooting' section in the INSTALL.md file first)  ***
***                                     ***
*****
c:\Variables\openssl-3.3.0>
```

Hình 8: Cấu hình trước khi cài đặt OpenSSL bằng Perl configure

Tại giao diện này sau khi thực hiện xong, **Perl** đã thông báo về các file đã được tạo ra gồm: **configdata.pm**, **makefile.in**, **makefile**

1.4. Sử dụng nmake để biên dịch mã nguồn của OpenSSL

nmake: Lệnh này sẽ biên dịch mã nguồn của **OpenSSL** thành các binary thực thi và thư viện cần thiết. Quá trình này sẽ tạo ra các file thực thi (như **openssl.exe** trên Windows) và các thư viện tĩnh hoặc động.



```
Administrator: x64 Native Tools Command Prompt for VS 2022 - nmake
*****
** Visual Studio 2022 Developer Command Prompt v17.9.6
** Copyright (c) 2022 Microsoft Corporation
*****
[vcvarsall.bat] Environment initialized for: 'x64'

C:\Windows\System32>cd /d c:\Variables\openssl-3.3.0

c:\Variables\openssl-3.3.0>nmake

Microsoft (R) Program Maintenance Utility Version 14.39.33523.0
Copyright (C) Microsoft Corporation. All rights reserved.

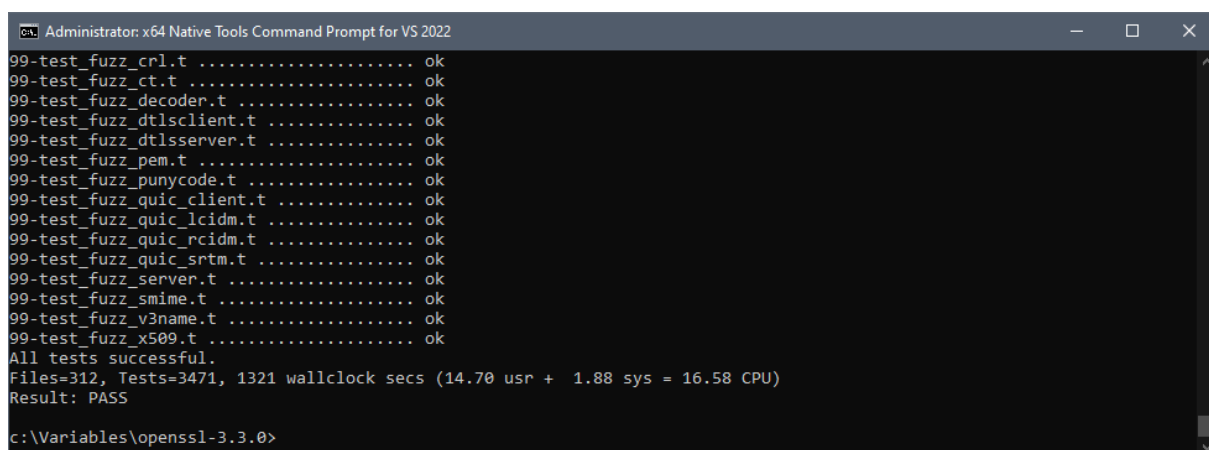
"C:\Strawberry\perl\bin\perl.exe" "util\mkinstallvars.pl" PREFIX=. BINDIR=apps LIBDIR= INCLUDEDIR=include APPLINKDIR=ms
ENGINESDIR=engines MODULESDIR=providers "VERSION=3.3.0" "LDLIBS=ws2_32.lib gdi32.lib advapi32.lib crypt32.lib user32.lib" >
builddata.pm
"C:\Strawberry\perl\bin\perl.exe" "-I." "-Mconfigdata" "-Mbuilddata" "util\dofile.pl" "-omakefile" "exporters\cmake\OpenSSLConfig.cmake.in" >
OpenSSLConfig.cmake
"C:\Strawberry\perl\bin\perl.exe" "-I." "-Mconfigdata" "-Mbuilddata" "util\dofile.pl" "-omakefile" "exporters\cmake\OpenSSLConfigVersion.cmake.in" >
OpenSSLConfigVersion.cmake
"C:\Strawberry\perl\bin\perl.exe" "-I." "-Iutil\perl" "-Mconfigdata" "-MOpenSSL::paramnames" "util\dofile.pl" "-omakefile" "crypto\params_idx.c.in" >
crypto\params_idx.c
"C:\Strawberry\perl\bin\perl.exe" "util\mkinstallvars.pl" "PREFIX=C:\OpenSSL-Win64" BINDIR=bin "LIBDIR=lib" INCLUDEDIR=include APPLINKDIR=include/openssl
"ENGINESDIR=C:\OpenSSL-Win64\lib\engines-3" "MODULESDIR=C:\OpenSSL-Win64\lib\o
```

Hình 9: Biên dịch mã nguồn OpenSSL trên Windows

Quá trình này sẽ mất tương đối thời gian, nhất là với hệ điều hành windows, bởi vì nó không hỗ trợ lệnh **make -j** như **Linux**.

nmake test: Sau khi biên dịch, ta có thể chạy các bài kiểm tra để đảm bảo rằng **OpenSSL** hoạt động đúng. Lệnh này sẽ thực hiện các bài kiểm tra được cung cấp kèm theo mã nguồn của **OpenSSL** để đảm bảo tính ổn định và độ tin cậy của mã nguồn.

Test thành công sẽ trông như thế này:



```
Administrator: x64 Native Tools Command Prompt for VS 2022

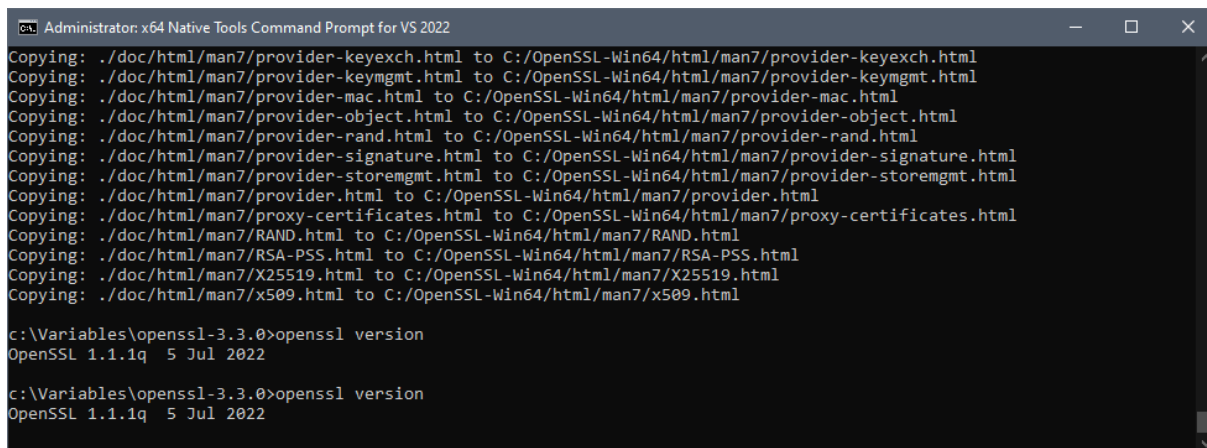
99-test_fuzz_crl.t ..... ok
99-test_fuzz_ct.t ..... ok
99-test_fuzz_decoder.t ..... ok
99-test_fuzz_dtlsclient.t ..... ok
99-test_fuzz_dtlsserver.t ..... ok
99-test_fuzz_pem.t ..... ok
99-test_fuzz_punycode.t ..... ok
99-test_fuzz_quic_client.t ..... ok
99-test_fuzz_quic_lcidm.t ..... ok
99-test_fuzz_quic_rcidm.t ..... ok
99-test_fuzz_quic_srtm.t ..... ok
99-test_fuzz_server.t ..... ok
99-test_fuzz_smime.t ..... ok
99-test_fuzz_v3name.t ..... ok
99-test_fuzz_x509.t ..... ok
All tests successful.
Files=312, Tests=3471, 1321 wallclock secs (14.70 usr + 1.88 sys = 16.58 CPU)
Result: PASS

c:\Variables\openssl-3.3.0>
```

Hình 10: Test trước khi cài đặt OpenSSL trên Windows

nmake install: Khi đã chắc chắn rằng OpenSSL đã được biên dịch và kiểm tra thành công, ta có thể cài đặt nó lên hệ thống của mình bằng lệnh này. Quá trình này sẽ sao chép các file thực thi và thư viện vào thư mục đã configure trước đó là:

C:\OpenSSL-Win64



```
Administrator: x64 Native Tools Command Prompt for VS 2022
Copying: ./doc/html/man7/provider-keyexch.html to C:/OpenSSL-Win64/html/man7/provider-keyexch.html
Copying: ./doc/html/man7/provider-keymgmt.html to C:/OpenSSL-Win64/html/man7/provider-keymgmt.html
Copying: ./doc/html/man7/provider-mac.html to C:/OpenSSL-Win64/html/man7/provider-mac.html
Copying: ./doc/html/man7/provider-object.html to C:/OpenSSL-Win64/html/man7/provider-object.html
Copying: ./doc/html/man7/provider-rand.html to C:/OpenSSL-Win64/html/man7/provider-rand.html
Copying: ./doc/html/man7/provider-signature.html to C:/OpenSSL-Win64/html/man7/provider-signature.html
Copying: ./doc/html/man7/provider-storemgmt.html to C:/OpenSSL-Win64/html/man7/provider-storemgmt.html
Copying: ./doc/html/man7/provider.html to C:/OpenSSL-Win64/html/man7/provider.html
Copying: ./doc/html/man7/proxy-certificates.html to C:/OpenSSL-Win64/html/man7/proxy-certificates.html
Copying: ./doc/html/man7/RAND.html to C:/OpenSSL-Win64/html/man7/RAND.html
Copying: ./doc/html/man7/RSA-PSS.html to C:/OpenSSL-Win64/html/man7/RSA-PSS.html
Copying: ./doc/html/man7/X25519.html to C:/OpenSSL-Win64/html/man7/X25519.html
Copying: ./doc/html/man7/x509.html to C:/OpenSSL-Win64/html/man7/x509.html

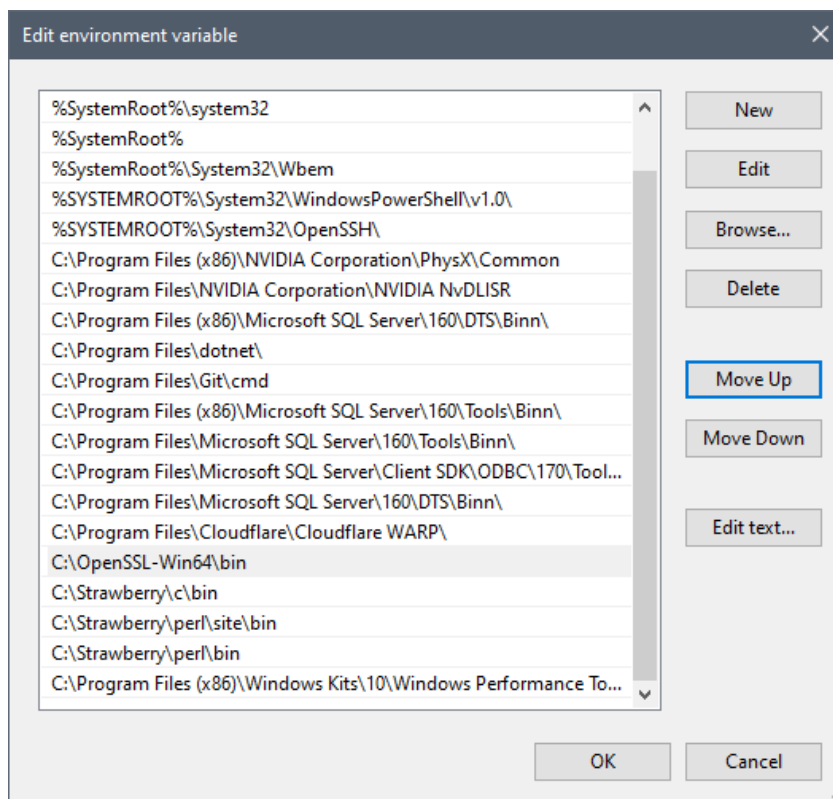
c:\Variables\openssl-3.3.0>openssl version
OpenSSL 1.1.1q 5 Jul 2022

c:\Variables\openssl-3.3.0>openssl version
OpenSSL 1.1.1q 5 Jul 2022
```

Hình 11: Cài đặt OpenSSL trên Windows

Ở đây sau khi cài đặt xong, mình đã thử check **openssl version**, thì windows đang nhận **OpenSSL** của thư viện của **Perl**.

Chúng ta sẽ thêm **OpenSSL** mới cài đặt vào [biến môi trường](#)^[tr.59], và đưa nó lên vị trí cao hơn **Perl**.



Hình 12: Cấu hình đường dẫn biến môi trường cho OpenSSL

1.5. Tích hợp OpenSSL vào phát triển chương trình C dùng Visual Studio Code

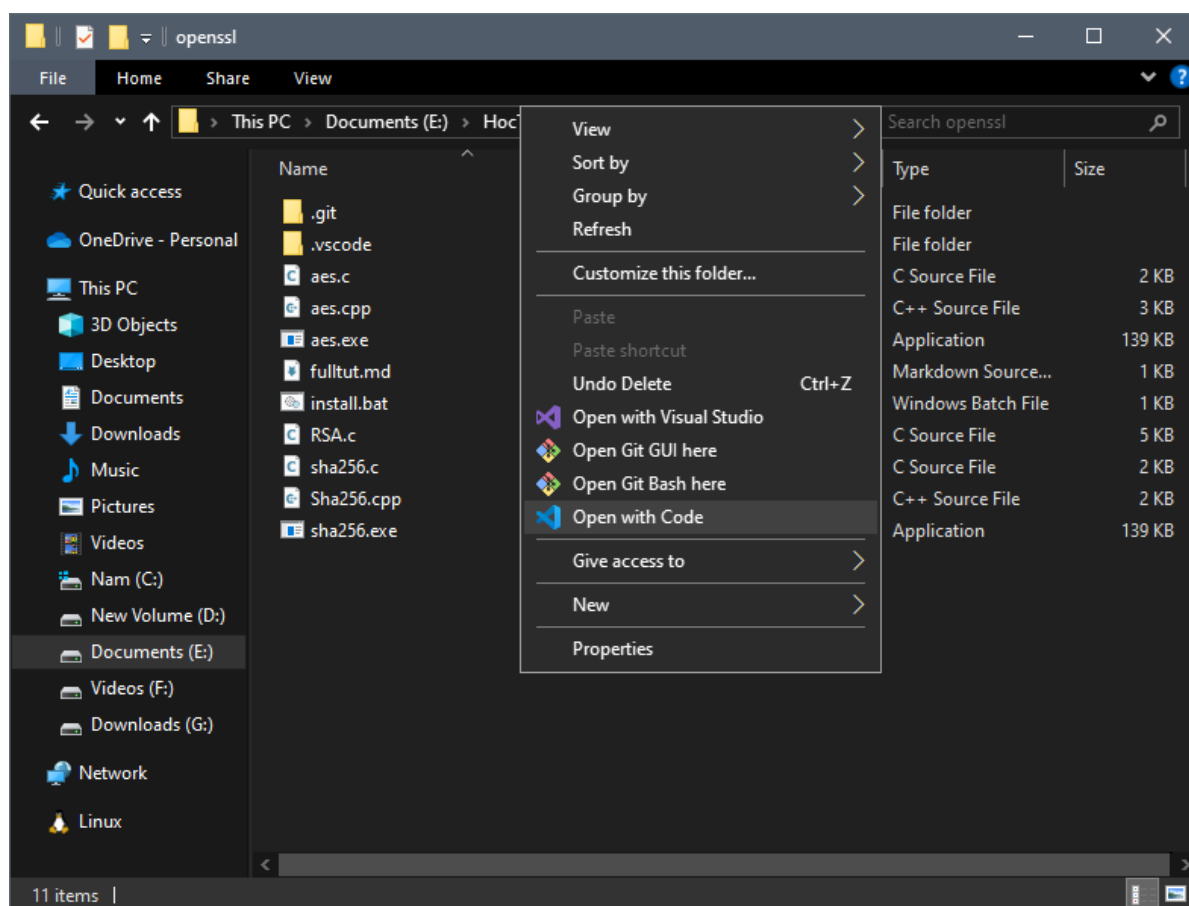
Việc tích hợp **OpenSSL** như được nhắc tới là đề cập đến việc:

- Tích hợp thư viện **lcrypto** và tích hợp các header của **OpenSSL** (nói cách khác là các file .c)
- Tích hợp các header vào **Intellisense** để xác định đối tượng của nó trong khi code để không bị lỗi văn bản.

Điều quan trọng khi làm việc với bất kì ngôn ngữ nào là xác định **Compiler** (trình biên dịch) và nếu cần **debug** thì sẽ phải quan tâm đến **Debugger** cho chương trình, vì vậy cần [cài đặt Compiler và Debugger](#)^[tr.53].

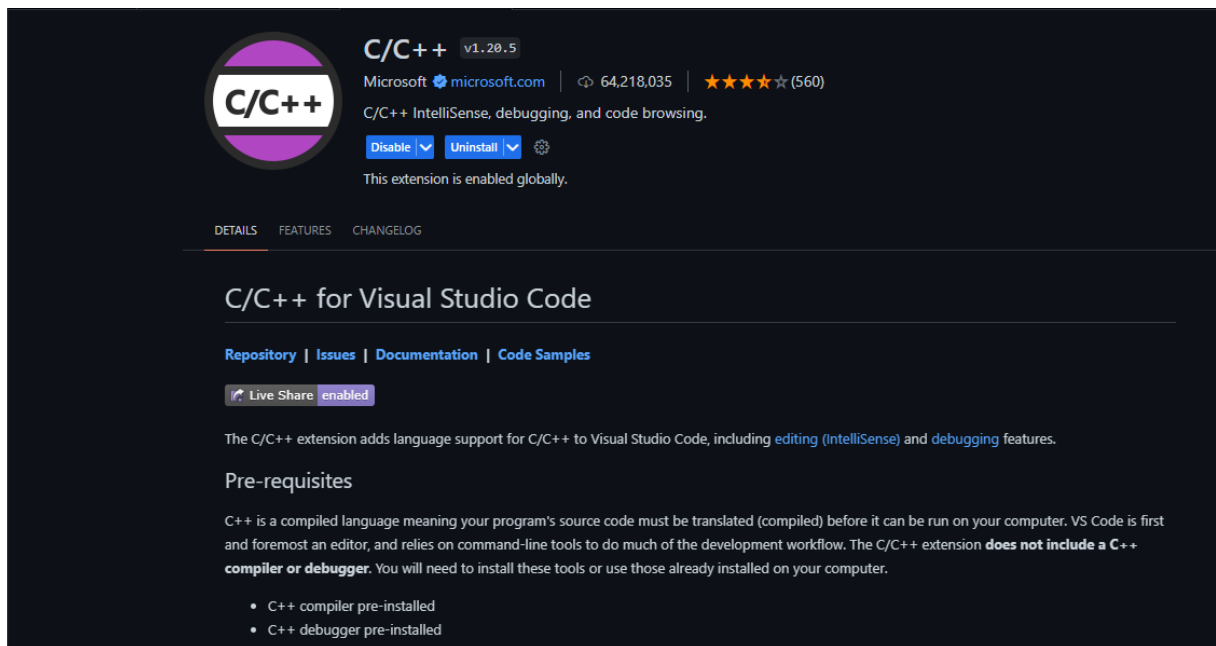
Sau đó, chúng ta cần tạo một thư mục làm việc (workspace) trong **Visual Code**.

Có nhiều cách để làm việc đó, đối với mình thì mình sẽ tạo một folder mới, đặt tên cho nó và sau đó chọn **Open With Code**.



Hình 13: Làm việc với C/C++ bằng Visual Studio Code

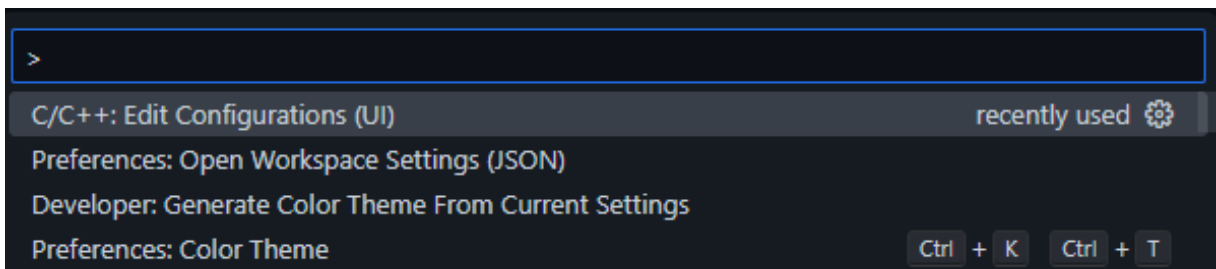
Ở trong WorkSpace đã mở này, điều đầu tiên cần làm là tải cái extension cần thiết cho Visual Code để compile, debug, hỗ trợ code (Intellisense) ngôn ngữ C/C++:



Hình 14: Extension để làm việc với C/C++

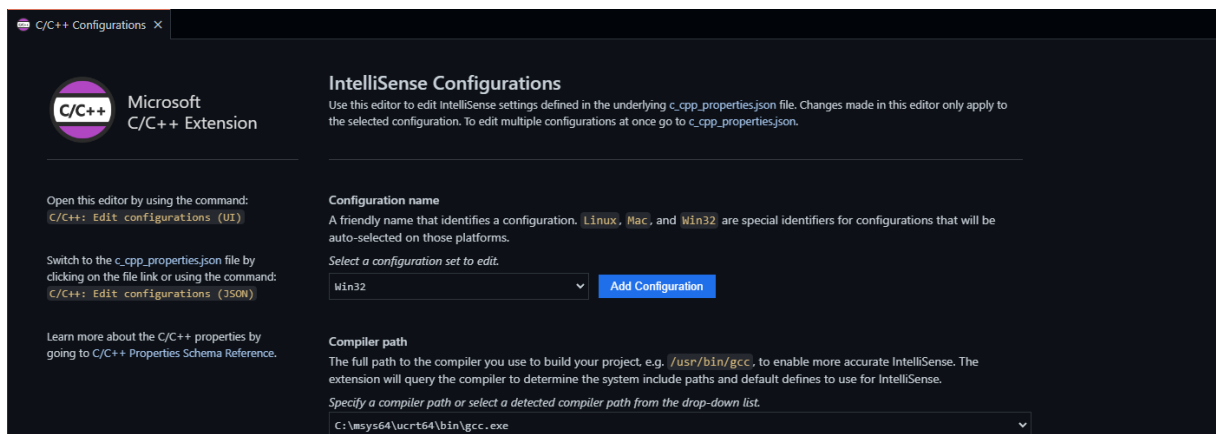
Sau đó chúng ta cần cấu hình **compiler** để **VS Code** hiểu chúng ta muốn sử dụng **compiler** nào trên môi trường của mình.

Để cấu hình nó, có thể sử dụng **Ctrl + Shift + P**, sau đó gõ “**C/C++: edit configurations (UI)**”



Hình 15: Cấu hình để làm việc với C/C++

Chọn lựa chọn **C/C++: edit configurations (UI)** sau đó sẽ có hộp thoại sau:



Hình 16: Cấu hình compiler path và include path

Tại đây chúng ta sẽ quan tâm 2 lựa chọn là, **Compiler path**, và **Include path**
Đối với mỗi ô này, lựa chọn đường dẫn hợp lý đến **compiler** và folder chứa header tương ứng.

- Trong trường hợp này compiler là **C:\msys64\ucrt\bin\gcc.exe**
- Và include path là: **C:\OpenSSL-Win64\include\openssl**

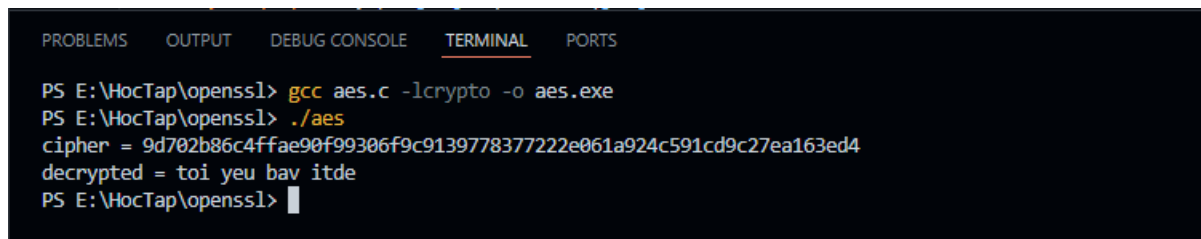
Tại bước này, chúng ta đã có thể tạm dừng và bắt đầu code với C. Và chúng ta có thể **compile** và **run** dùng terminal (command prompt hoặc powershell đều được):

Bước 1: Mở **Intergrated Terminal** (hoặc Mở **Terminal** bình thường ta vẫn dùng rồi di chuyển đến folder WorkSpace)

Bước 2: Thực hiện lệnh: gcc [tên file compile].c -lcrypto -o[tên output].exe

- **Đối với PowerShell**

Bước 3: Thực hiện lệnh: ./[tên output]

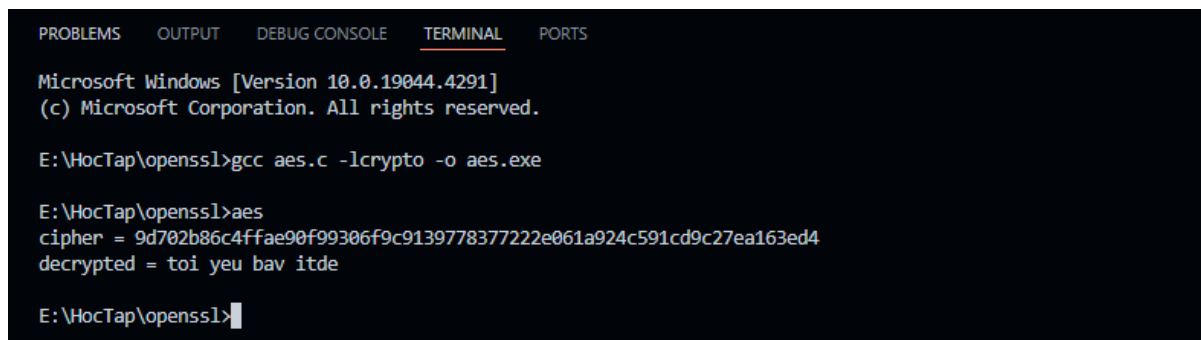


```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS E:\HocTap\openssl> gcc aes.c -lcrypto -o aes.exe
PS E:\HocTap\openssl> ./aes
cipher = 9d702b86c4ffae90f99306f9c913977837722e061a924c591cd9c27ea163ed4
decrypted = toi yeu bav itde
PS E:\HocTap\openssl> |
```

Hình 17: Biên dịch và chạy bằng PowerShell

- **Đối với Command Prompt**

Bước 3: Thực hiện lệnh: [tên output]



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Microsoft Windows [Version 10.0.19044.4291]
(c) Microsoft Corporation. All rights reserved.

E:\HocTap\openssl>gcc aes.c -lcrypto -o aes.exe

E:\HocTap\openssl>aes
cipher = 9d702b86c4ffae90f99306f9c913977837722e061a924c591cd9c27ea163ed4
decrypted = toi yeu bav itde

E:\HocTap\openssl>|
```

Hình 18: Biên dịch và chạy bằng Command Prompt

1.6. Làm việc với nhiều file mã nguồn và sử dụng debug & run trong VS Code

Bước 1: Cấu hình file `c_cpp_properties.json`

```
.vscode > {} c_cpp_properties.json > ...
1  {
2      "configurations": [
3          {
4              "name": "Win32",
5              "includePath": [
6                  "${workspaceFolder}/**",
7                  "C:\\OpenSSL-Win64\\include\\openssl"
8              ],
9              "defines": [
10                 "_DEBUG",
11                 "UNICODE",
12                 "_UNICODE"
13             ],
14             "cStandard": "c17",
15             "cppStandard": "gnu++17",
16             "intelliSenseMode": "windows-gcc-x64",
17             "compilerPath": "C:\\msys64\\ucrt64\\bin\\gcc.exe"
18         }
19     ],
20     "version": 4
21 }
```

Hình 19: Cấu hình file `c_cpp_properties.json`

Bước này đã được thực hiện từ trước, khi chúng ta thực hiện C/C++: **edit configurations (UI)**, chỉ là đây là dạng Json của nó.

Bước 2: Cấu hình file `launch.json` (debug)

```
.vscode > {} launch.json > ...
1  {
2      "version": "0.2.0",
3      "configurations": [
4          {
5              "name": "g++.exe - Build and debug active file",
6              "type": "cppdbg",
7              "request": "launch",
8              "program": "${fileDirname}\\${fileBasenameNoExtension}.exe",
9              "args": [],
10             "stopAtEntry": false,
11             "cwd": "${fileDirname}",
12             "environment": [],
13             "externalConsole": false,
14             "MIMode": "gdb",
15             "miDebuggerPath": "C:\\msys64\\ucrt64\\bin\\gdb.exe", // Sửa đường dẫn đến gdb
16             "setupCommands": [
17                 {
18                     "description": "Enable pretty-printing for gdb",
19                     "text": "-enable-pretty-printing",
20                     "ignoreFailures": true
21                 }
22             ],
23             "preLaunchTask": "C/C++: g++.exe build active file" // Sửa tên task ở đây
24         }
25     ]
26 }
```

Hình 20: Cấu hình file `launch.json`

Ở bước này chúng ta sẽ dành sự chú ý cho giá trị của **miDebuggerPath**, giá trị của nó sẽ là đường dẫn đến debugger, ở đây là **C:\msys64\ucrt64\bin\gdb.exe**

Các dòng khác có thể được thiết lập tự động bằng nút “add configurations” của VS Code.

Bước 3: Cấu hình file **tasks.json** (build)



```
.vscode > {} tasks.json > ...
1  {
2    "tasks": [
3      {
4        "type": "cppbuild",
5        "label": "C/C++: g++.exe build active file",
6        "command": "C:\\msys64\\ucrt64\\bin\\gcc.exe",
7        "args": [
8          "-g",
9          "${file}",
10         "-lcrypto", // Thêm option để chỉ định thư viện lcrypto
11         "-o",
12         "${fileDirname}\\${fileBasenameNoExtension}.exe",
13         "-Ic:\\OpenSSL-Win64\\include\\openssl"
14       ],
15       "options": {
16         "cwd": "${workspaceFolder}"
17       },
18       "problemMatcher": [
19         "$gcc"
20       ],
21       "group": {
22         "kind": "build",
23         "isDefault": true
24       },
25       "detail": "Task generated by Debugger."
26     },
27   ],
28   "version": "2.0.0"
29 }
```

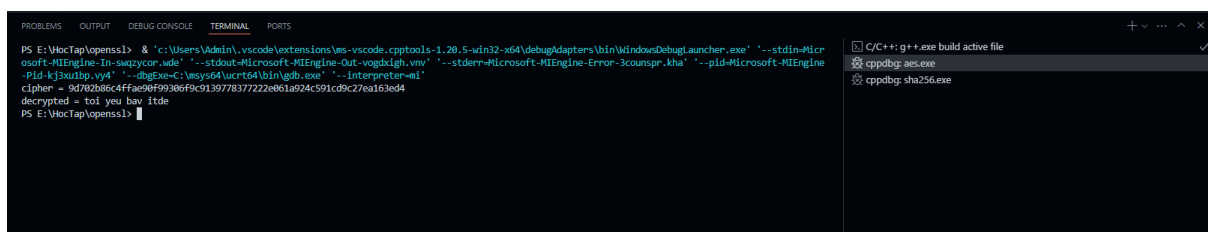
Hình 21: Cấu hình **tasks.json**

Tại đây, quan tâm chính đến phần **command** và **args**:

Command là chuỗi câu lệnh đầu

Args là các tham số để đưa vào câu lệnh:

- **-g**: xác định **compiler**
- **\${file}**: tên file được biên dịch
- **-lcrypto**: xác định thư viện mở rộng muốn được thêm vào khi biên dịch, ở đây là thư viện crypto và được windows công nhận chính thức bằng **-lcrypto**, đối với những thư viện không được công nhận, phải chỉ rõ đường dẫn: **-L{đường dẫn}**
- **-o**: xác định output
- **\${fileDirname}\\\${fileBasenameNoExtension}.exe**: đặt tên và extension cho file.
- **-Ic:\\OpenSSL-Win64\\include\\openssl** : là đường dẫn đến các file header.



Có thể thấy mình đã thực hiện **debug and run** 2 file trong cùng WorkSpace.

Có nghĩa là chương trình đã được build thành tệp thực thi và được gắn debug trong quá trình build, để gọi lại chương trình một lần nữa, có thể gọi đến file thực thi như cách đã thực hiện ở phần trước.

KẾT LUẬN:

Việc biên dịch và build **OpenSSL** từ mã nguồn trên windows quá phức tạp, yêu cầu quá nhiều các tiên quyết, mất rất nhiều thời gian để thiết lập. Hơn nữa, để tích hợp các thư viện và header của **OpenSSL** vào quá trình phát triển cũng phức tạp hơn nhất là về mặt câu lệnh trong giao diện dòng lệnh.

Nhìn vào 2 điểm này để thấy rõ được điểm yếu của hệ điều hành **Windows**, nhất là trong việc xây dựng và phát triển ứng dụng/ phần mềm.

Chính vì thế, ở phần tiếp theo, chúng mình sẽ đi tìm hiểu về việc biên dịch và build **OpenSSL** trên **Linux**, tích hợp nó vào môi trường phát triển như cách ta đã làm với **Windows**, nhưng nó sẽ đơn giản và tiết kiệm thời gian hơn.

2. Xây dựng và cài đặt OpenSSL cho Linux (Ubuntu)

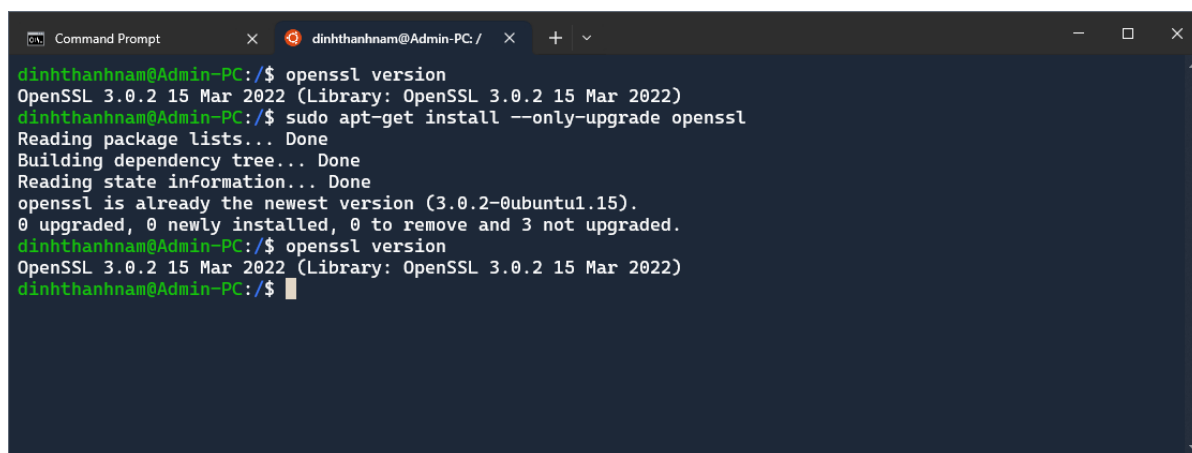
Có nhiều cách để cài đặt hệ điều hành **Linux**, trong đó, **Ubuntu** là 1 **Linux Distribution (distro)**⁴.

Trong bài báo cáo này chúng mình sẽ sử dụng tính năng hỗ trợ bởi Window để cài đặt một hệ điều hành **Linux** sử dụng ảo hoá, là [Windows subsystem for linux](#)^[tr.56].

2.1. Kiểm tra phiên bản OpenSSL

Kiểm tra phiên bản bằng lệnh: `openssl version`

Thử cài bản mới hơn bằng: `apt-get install --only-upgrade openssl`



```
dinhthanhnam@Admin-PC:/$ openssl version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
dinhthanhnam@Admin-PC:/$ sudo apt-get install --only-upgrade openssl
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openssl is already the newest version (3.0.2-0ubuntu1.15).
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
dinhthanhnam@Admin-PC:/$ openssl version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
dinhthanhnam@Admin-PC:/$
```

Hình 22: Kiểm tra OpenSSL trên Linux

Tại đây chúng ta thấy rõ phiên bản hiện tại không phải là phiên bản mới nhất, lý do là vì **Linux** có rất nhiều tài nguyên phụ thuộc vào **OpenSSL**, vậy nên phiên bản mặc định sẽ là phiên bản ổn định nhất cho **Linux**.

Tuy nhiên, điều đó không ảnh hưởng đến việc cài đặt một bản **OpenSSL** mới hơn, nhất là khi chúng ta cần sử dụng phiên bản nhất định nào đó cho việc phát triển của mình.

2.2. Tải và giải nén mã nguồn của OpenSSL

Chúng mình sẽ tải mã nguồn của bản **OpenSSL** giống với khi thực hiện trên **Windows**.

Copy link tải xuống từ trang chính thức của OpenSSL rồi thực hiện lệnh:

`Wget [URL]`

```
dinhthanhnam@Admin-PC: ~$ pwd
/
dinhthanhnam@Admin-PC:~$ cd /home/dinhthanhnam/
dinhthanhnam@Admin-PC:~$ wget https://www.openssl.org/source/openssl-3.3.0.tar.gz
--2024-05-11 19:26:35-- https://www.openssl.org/source/openssl-3.3.0.tar.gz
Resolving www.openssl.org (www.openssl.org)... 34.36.58.177, 2600:1901:0:1812::
Connecting to www.openssl.org (www.openssl.org)|34.36.58.177|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18038030 (17M) [application/tar+gzip]
Saving to: 'openssl-3.3.0.tar.gz'

openssl-3.3.0.tar.gz      100%[=====>] 17.20M  6.99MB/s   in 2.5s
2024-05-11 19:26:38 (6.99 MB/s) - 'openssl-3.3.0.tar.gz' saved [18038030/18038030]

dinhthanhnam@Admin-PC:~$
```

Hình 24: Tải xuống mã nguồn của OpenSSL

Sau đó thực hiện giải nén: tar xf [tên tệp nén]

```
dinhthanhnam@Admin-PC: ~$ pwd
/
dinhthanhnam@Admin-PC:~$ cd /home/dinhthanhnam/
dinhthanhnam@Admin-PC:~$ wget https://www.openssl.org/source/openssl-3.3.0.tar.gz
--2024-05-11 19:26:35-- https://www.openssl.org/source/openssl-3.3.0.tar.gz
Resolving www.openssl.org (www.openssl.org)... 34.36.58.177, 2600:1901:0:1812::
Connecting to www.openssl.org (www.openssl.org)|34.36.58.177|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 18038030 (17M) [application/tar+gzip]
Saving to: 'openssl-3.3.0.tar.gz'

openssl-3.3.0.tar.gz      100%[=====>] 17.20M  6.99MB/s   in 2.5s
2024-05-11 19:26:38 (6.99 MB/s) - 'openssl-3.3.0.tar.gz' saved [18038030/18038030]

dinhthanhnam@Admin-PC:~$
```

Hình 23: Giải nén mã nguồn của OpenSSL

2.3. Chạy file cấu hình bằng lệnh ./config

Ở đây, như được nhắc đến, chúng mình sẽ chạy file cấu hình có sẵn trong mã nguồn của OpenSSL, là file **config**, chúng ta cần sử dụng **Perl** như windows để cấu hình file này, mà chỉ cần dùng lệnh: `./[tên file]`

Các bước thực hiện sẽ gồm: di chuyển đến folder chứa mã nguồn và chứa file **config** để chạy.

```
dinhthanhnam@Admin-PC: ~$ cd openssl-3.3.0/
dinhthanhnam@Admin-PC:~/openssl-3.3.0$ ./config
Configuring OpenSSL version 3.3.0 for target linux-x86_64
Using os-specific seed configuration
Created configdata.pm
Running configdata.pm
Created Makefile.in
Created Makefile
Created include/openssl/configuration.h

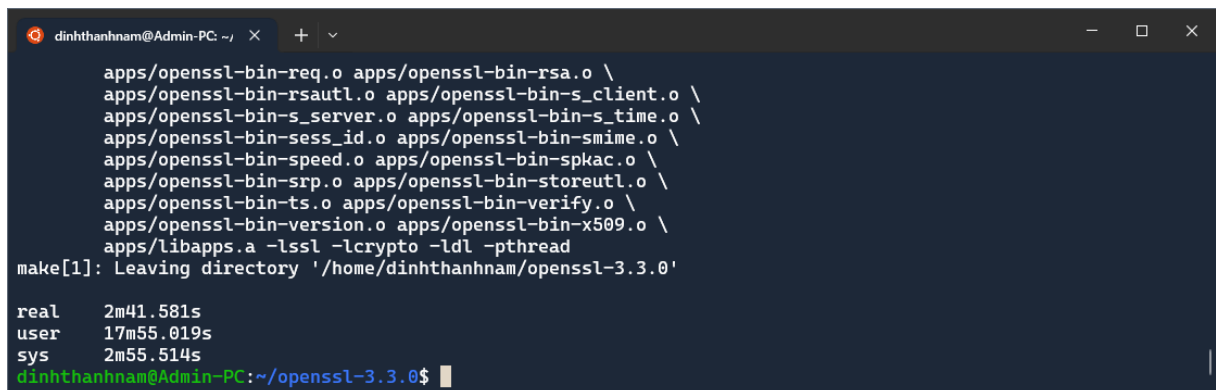
*****
```

Hình 25: Cấu hình trước khi biên dịch và cài đặt OpenSSL

2.4. Biên dịch và cài đặt OpenSSL

Tại cùng folder chứa mã nguồn như trên, chúng mình có thể dùng lệnh **make** để bắt đầu biên dịch.

Tuy nhiên, **Linux** có hỗ trợ thêm lệnh **make -j** để biên dịch nhanh hơn, sử dụng toàn bộ luồng của CPU, và thực tế, quá trình biên dịch này là quá trình mất nhiều thời gian nhất, chúng mình sẽ thử dùng lệnh **make -j** để xem nó có thể nhanh hơn biên dịch trên Windows tới bao nhiêu %. Mình sẽ thêm 1 tham số **time** vào trước để đo thời gian chính xác.



```
dinhthanhnam@Admin-PC: ~/openssl-3.3.0
apps/openssl-bin-req.o apps/openssl-bin-rsa.o \
apps/openssl-bin-rsautil.o apps/openssl-bin-s_client.o \
apps/openssl-bin-s_server.o apps/openssl-bin-s_time.o \
apps/openssl-bin-sess_id.o apps/openssl-bin-smime.o \
apps/openssl-bin-speed.o apps/openssl-bin-spkac.o \
apps/openssl-bin-srp.o apps/openssl-bin-storeutl.o \
apps/openssl-bin-ts.o apps/openssl-bin-verify.o \
apps/openssl-bin-version.o apps/openssl-bin-x509.o \
apps/libapps.a -lssl -lcrypto -ldl -pthread
make[1]: Leaving directory '/home/dinhthanhnam/openssl-3.3.0'

real    2m41.581s
user    17m55.019s
sys     2m55.514s
dinhthanhnam@Admin-PC: ~/openssl-3.3.0$
```

Hình 26: Biên dịch mã nguồn OpenSSL trên Linux

Thời gian đo được là 2 phút 41 giây. Và thời gian mình đã thực hiện biên dịch trên Windows là hơn 30 phút, vậy có thể nói là nhanh hơn 1000% trên **Linux**.

Không chỉ biên dịch diễn ra nhanh hơn, mà cả những bước khác như make test, make install cũng nhanh hơn.

Tương tự như Windows, mình sẽ chạy 2 lệnh tiếp theo là:

make test

sudo make install

Với điều kiện là **make test** thành công, thì chắc chắn **sudo make install** sẽ thành công, chỉ cần chờ một lúc cho quá trình cài đặt xong.

2.5. Cấu hình để sử dụng phiên bản OpenSSL mới nhất và thư viện của nó

Sau khi đã thực hiện cài đặt **OpenSSL** bản mới nhất, vẫn thấy rằng Linux đang sử dụng OpenSSL cũ, điều này là do ta chưa cấu hình cho Linux sử dụng OpenSSL bản mới, và chưa cấu hình thư viện cho nó:

```
dinhthanhnam@Admin-PC: /  x + v
dinhthanhnam@Admin-PC:/$ openssl version
OpenSSL 3.0.2 15 Mar 2022 (Library: OpenSSL 3.0.2 15 Mar 2022)
dinhthanhnam@Admin-PC:/$
```

Hình 28: Kiểm tra phiên bản OpenSSL đang được sử dụng trên Linux

Chúng mình sẽ tạo ra một file openssl.conf vào thư mục **/etc/ld.so.conf.d**

```
dinhthanhnam@Admin-PC: /  x + v
dinhthanhnam@Admin-PC:/$ sudo nano /etc/ld.so.conf.d/openssl.conf
```

Hình 27: Cấu hình cho Linux sử dụng bản OpenSSL mong muốn(1)

Ghi đường dẫn **OpenSSL** mới cài vào đó, tương đương: **/usr/local/bin**

Và đường dẫn đến thư viện **libcrypto** và **libssl**, tương đương: **/usr/local/lib64**

```
dinhthanhnam@Admin-PC: /  x + v
GNU nano 6.2 /etc/ld.so.conf.d/openssl.conf
/usr/local/bin
/usr/local/lib64

^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location   M-U Undo
^X Exit      ^R Read File  ^_ Replace    ^U Paste      ^J Justify    ^_ Go To Line M-E Redo
```

Hình 29: Cấu hình cho Linux sử dụng bản OpenSSL mong muốn(2)

Lưu rồi sau đó thực hiện **sudo ldconfig** để cập nhật

```
dinhthanhnam@Admin-PC: /  x + v
dinhthanhnam@Admin-PC:/$ sudo nano /etc/ld.so.conf.d/openssl.conf
dinhthanhnam@Admin-PC:/$ sudo ldconfig
/sbin/ldconfig.real: /usr/lib/wsl/lib/libcuda.so.1 is not a symbolic link

dinhthanhnam@Admin-PC:/$
```

Hình 30: Cấu hình cho Linux sử dụng bản OpenSSL mong muốn(3)

Kiểm tra lại xem đã đúng chưa:

```
dinhthanhnam@Admin-PC: /usr/local/lib64$ cd /
dinhthanhnam@Admin-PC:/$ sudo nano /etc/ld.so.conf.d/openssl.conf
dinhthanhnam@Admin-PC:/$ sudo ldconfig
/sbin/ldconfig.real: /usr/lib/wsl/lib/libcudart.so.1 is not a symbolic link

dinhthanhnam@Admin-PC:/$ openssl version
OpenSSL 3.3.0 9 Apr 2024 (Library: OpenSSL 3.3.0 9 Apr 2024)
dinhthanhnam@Admin-PC:/$
```

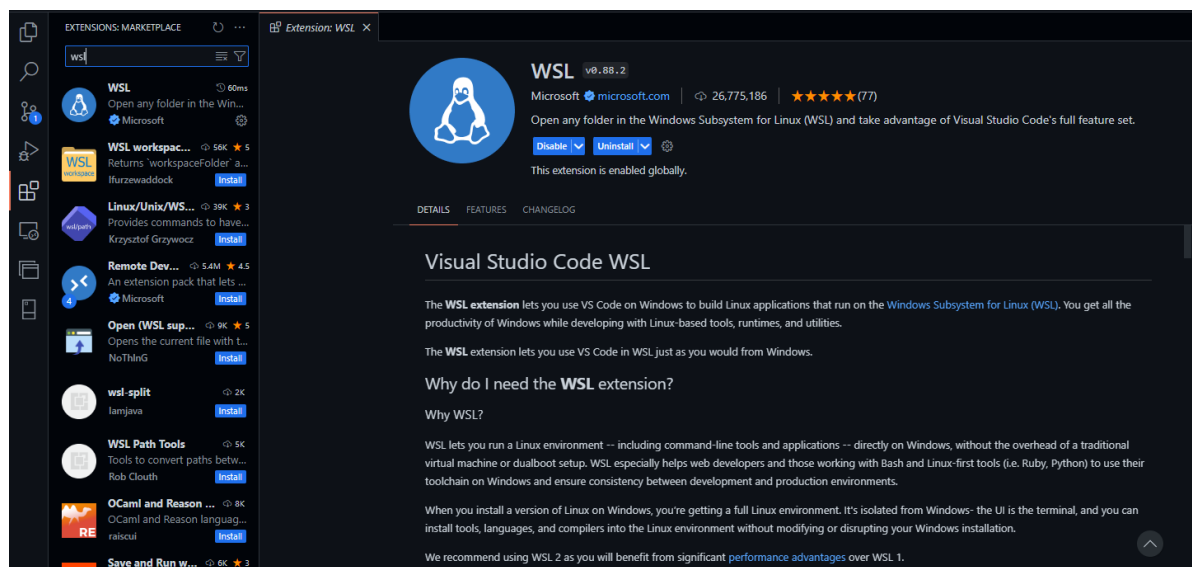
Hình 31: Kiểm tra lại phiên bản OpenSSL sau khi đã thiết lập lại

2.6. Làm việc với ngôn ngữ C trên Linux có sử dụng thư viện của OpenSSL

Để làm việc với ngôn ngữ C, editor mặc định của nó thường là **gedit**, có thể cài bằng **sudo apt-get install gedit**.

Tuy nhiên, chúng ta sẽ bỏ qua nó, và làm việc bằng **Visual Studio Code** của chúng ta trên Windows luôn.

Để làm được, chúng ta sẽ cần cài đặt một extension cho **VS Code**:



Hình 32: Cài đặt WSL cho Visual Studio Code

Sau khi cài đặt extension này, chúng ta có thể làm việc với thư mục WorkSpace trên **WSL** như mà một Editor của **Linux**.

Để bắt đầu làm việc với C và thư viện **OpenSSL** trên **Linux**, chúng ta sẽ thực hiện tương tự như trên **Windows**.

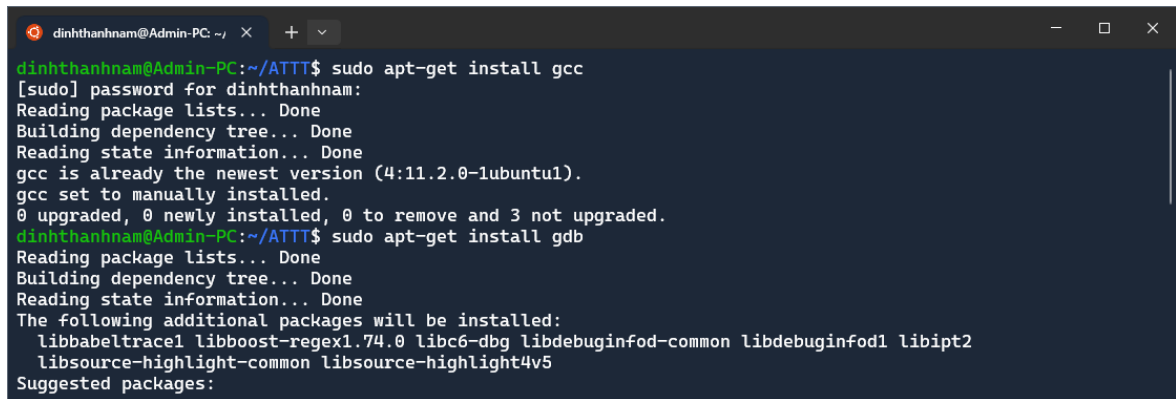
Chúng ta sẽ phải cài **C/C++ Extension** cho **WSL** tương tự như với **Windows**

Cài đặt công cụ **compile** và **debug** dễ dàng bằng **apt-get install**, không cần phức tạp như trên Windows.

Ở đây chúng ta thực hiện cài **gcc** và **gdb** như sau:

Sudo apt-get install gcc

Sudo apt-get install gdb



```
dinhthanhnam@Admin-PC: ~/ATT$ sudo apt-get install gcc
[sudo] password for dinhthanhnam:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
gcc is already the newest version (4:11.2.0-1ubuntu1).
gcc set to manually installed.
0 upgraded, 0 newly installed, 0 to remove and 3 not upgraded.
dinhthanhnam@Admin-PC: ~/ATT$ sudo apt-get install gdb
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  libbabeltrace1 libboost-regex1.74.0 libc6-dbg libdebuginfod-common libdebuginfod1 libipt2
  libsource-highlight-common libsource-highlight4v5
Suggested packages:
```

Hình 33: Sử dụng **apt-get** để cài đặt trên Linux

Vị trí cài đặt của **gdb** và **gcc** ở thư mục: **/usr/bin/**, sau đó chúng ta có chỉnh sửa đường dẫn trong **launch.json** và **tasks.json** hoàn toàn tương tự như thực hiện trên Windows.

Điều duy nhất khác là **terminal** của chúng ta sẽ là giao diện bash của **Ubuntu** luôn:



Hình 34: Giao diện **bash Ubuntu** tích hợp trong **Visual Studio Code**

Điều này tạo ra sự tiện lợi vô cùng lớn cho lập trình viên trong quá trình làm việc.

KẾT LUẬN: Tổng quan, mặc dù cả **Windows** và **Linux** đều có thể làm việc tốt với mã nguồn C và thư viện **OpenSSL**, nhưng **Linux** mang lại một trải nghiệm phát triển linh hoạt và thuận tiện hơn. Mặc dù **Windows** cung cấp một môi trường giao diện người dùng tốt, nhưng việc cài đặt và quản lý thư viện lại vô cùng phức tạp. Trong khi đó, **Linux** vừa cung cấp một môi trường phát triển mạnh mẽ với các công cụ dòng lệnh vừa có hệ thống quản lý gói, giúp cho việc phát triển và tích hợp phần mềm trở nên dễ dàng và linh hoạt hơn.

ĐIỀU KIỆN TIÊN QUYẾT

1. Strawberry Perl



Hình 35: Logo Strawberry Perl

1.1. Giới thiệu

Strawberry Perl là một bản phân phối Perl mã nguồn mở dành cho Windows, bao gồm trình biên dịch C/C++ và Mingw-w64 với các thư viện đi kèm. **Strawberry Perl** đi kèm với các công cụ phát triển phần mềm phổ biến và đã được cấu hình sẵn, giúp người dùng cài đặt Perl một cách dễ dàng. Điều này phân biệt Strawberry Perl với các bản phân phối Perl khác, ảnh hưởng đến các bản phân phối khác như ActivePerl.

1.2. Lý do cần thiết

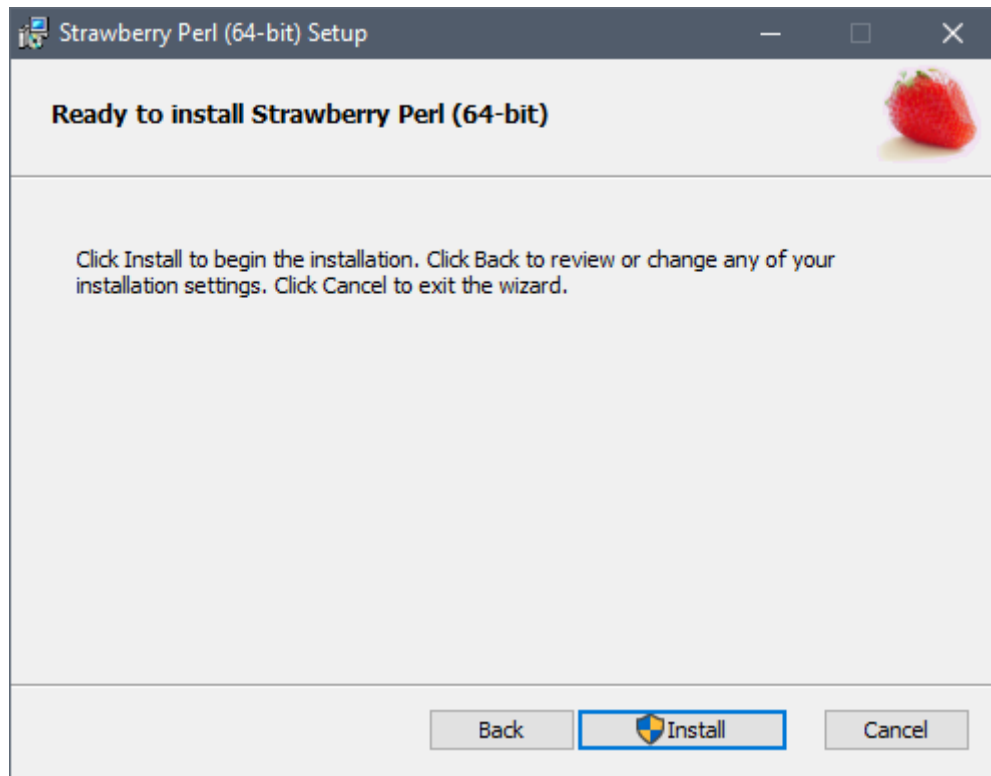
Phải kể đến là **Strawberry Perl** cung cấp trình biên dịch C/C++ cùng thư viện đi kèm của Mingw-w64.

Hơn cả, **Strawberry Perl** cung cấp lệnh cần thiết để tạo file cấu hình mã nguồn trước khi đưa vào biên dịch bằng câu lệnh:

```
perl configure
```


1.3. Tải và cài đặt

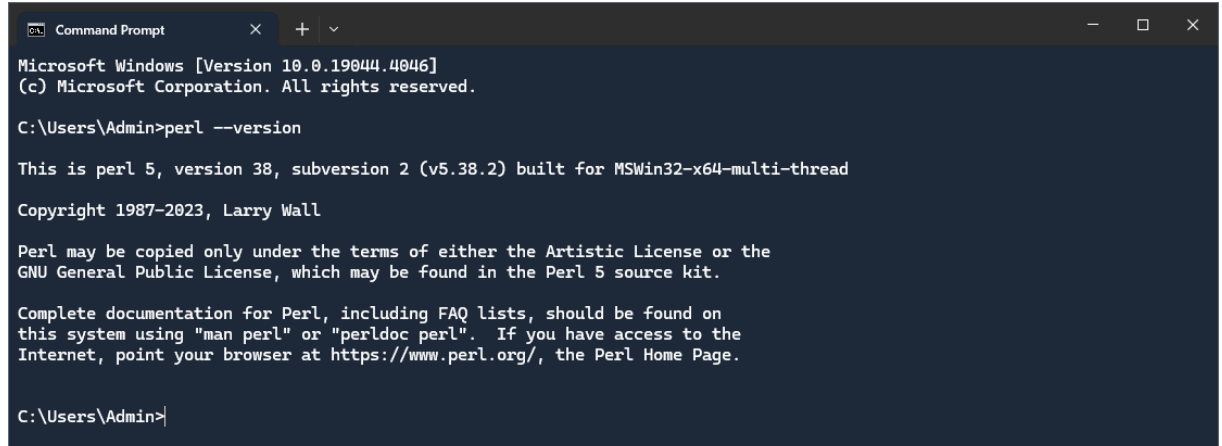
Tải và cài đặt **Strawberry Perl** như những ứng dụng thông thường tại trang chính thức của nó: <https://strawberryperl.com/>



Hình 36: Tải và cài đặt Strawberry Perl trên Windows

Sau khi cài đặt **Perl**, đảm bảo môi trường đã nhận **Perl**, mặc định là **Perl** đã tự thêm vào môi trường bằng quyền Admin tuy nhiên, vẫn có thể kiểm tra lại bằng câu lệnh sau trong **Terminal** hoặc thực hiện [kiểm tra biến môi trường](#)^[tr.59]:

```
perl --version
```



Hình 37: Kiểm tra cài đặt thành công Perl

2. NASM (Netwide Assembler)



Hình 38: Logo NASM

2.1. Giới thiệu

NASM (Netwide Assembler) là một bộ assembler mã nguồn mở, được sử dụng để dịch mã assembly thành mã máy cho nhiều kiến trúc CPU khác nhau.

2.2. Lý do cần thiết

Dịch mã Assembly:

NASM cho phép biên dịch mã assembly thành mã máy tương ứng cho kiến trúc CPU mục tiêu. Điều này cho phép viết mã assembly để tận dụng các tính năng và hiệu suất tối ưu của CPU một cách chi tiết và linh hoạt.

Hỗ trợ đa kiến trúc:

NASM hỗ trợ nhiều kiến trúc CPU, bao gồm x86 (32-bit và 64-bit), x86-64 (AMD64 và Intel 64), và nhiều kiến trúc khác như ARM và MIPS. Điều này cho phép viết mã assembly độc lập với kiến trúc, giúp tối ưu hóa mã cho nhiều nền tảng.

Tính di động:

NASM có thể được sử dụng trên nhiều hệ điều hành, bao gồm Windows, Linux và macOS. Điều này cho phép bạn xây dựng và phát triển ứng dụng trên nhiều nền tảng mà không cần phải thay đổi mã nguồn.

Tích hợp Với VC++ 64-bit:

Trong quá trình build và cài đặt bằng **Visual C++ 64-bit**, **NASM** thường được sử dụng để biên dịch mã assembly hoặc mã nguồn tương ứng thành mã máy.

Hiệu suất và kiểm soát:

Việc viết mã assembly cho phép kiểm soát hoàn toàn việc tối ưu hóa hiệu suất và sử dụng các tính năng đặc biệt của CPU một cách tối ưu nhất. **NASM** là công cụ giúp làm điều này một cách dễ dàng và linh hoạt.

















2.3. Tải và cài đặt

Mặc định, sau khi đã [cài đặt gói Phát triển với C++ của Visual Studio](#) [tr.51], **NASM** đã được tích hợp sẵn trong các môi trường phát triển VC++

Tuy nhiên, chúng mình hoàn toàn có thể cài đặt **NASM** bản mới hơn nếu đó là nhu cầu sử dụng, tại:




<https://nasm.us/>

Chọn phiên bản cài đặt phù hợp, ở đây là Win64:

Index of /pub/nasm/releasebuilds/2.16.03				
	Name	Last modified	Size	Description
	Parent Directory		-	
	doc/	2024-04-17 10:06	-	Online documentation
	dos/	2024-04-17 10:08	-	MS-DOS executables
	linux/	2024-04-17 10:08	-	Linux packages
	macosx/	2024-04-17 10:08	-	MacOS X packages
	win32/	2024-04-17 10:08	-	Windows packages (32 bit)
	win64/	2024-04-17 10:08	-	Windows packages (64 bit)
	git.id	2024-04-17 10:08	41	Corresponding git revision ID
	nasm-2.16.03-xdoc.tar.bz2	2024-04-17 10:06	947K	Downloadable documentation
	nasm-2.16.03-xdoc.tar.gz	2024-04-17 10:06	1.0M	Downloadable documentation
	nasm-2.16.03-xdoc.tar.xz	2024-04-17 10:06	853K	Downloadable documentation
	nasm-2.16.03-xdoc.zip	2024-04-17 10:06	1.0M	Downloadable documentation
	nasm-2.16.03.tar.bz2	2024-04-17 10:05	1.3M	Source code
	nasm-2.16.03.tar.gz	2024-04-17 10:05	1.6M	Source code
	nasm-2.16.03.tar.xz	2024-04-17 10:05	1.0M	Source code
	nasm-2.16.03.zip	2024-04-17 10:05	1.9M	Source code
Browse source code for this build				

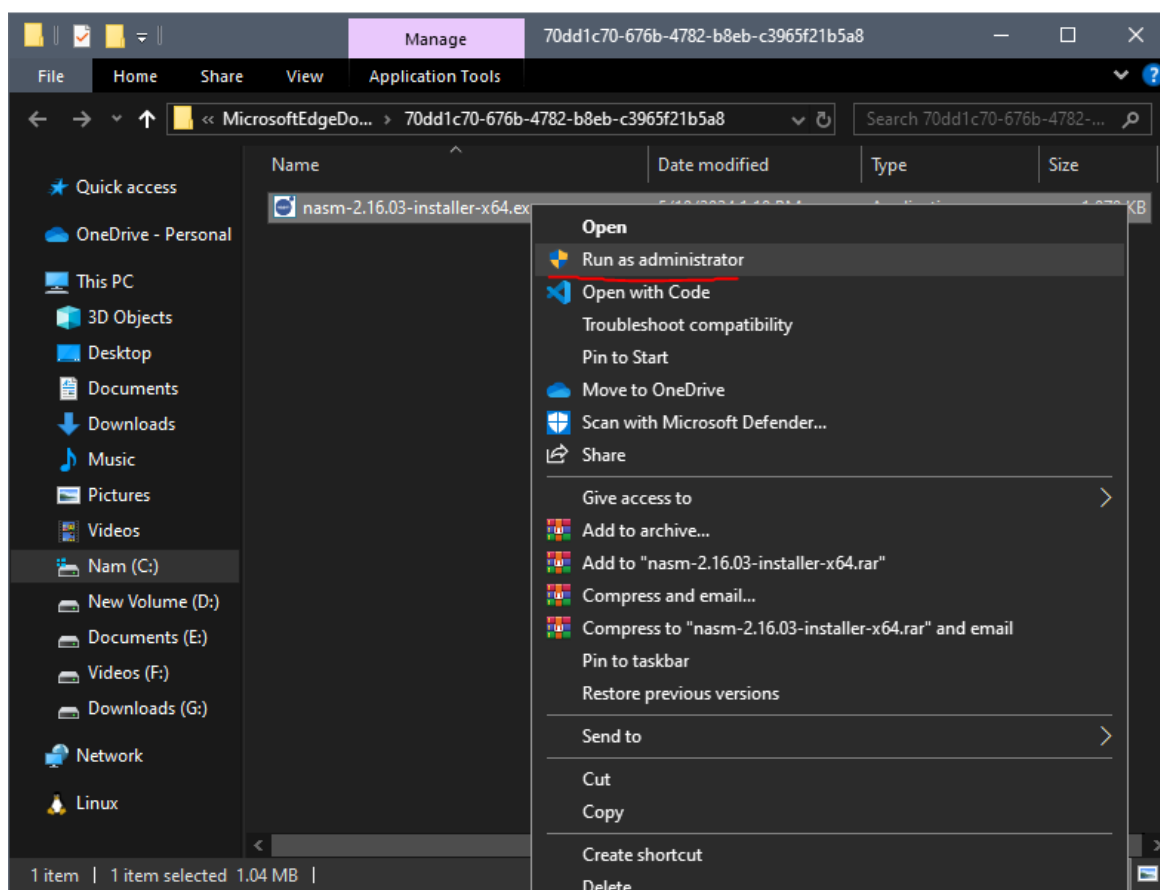
Hình 39: Tải và cài đặt NASM (1)

Chọn tải file exe hoặc zip tùy nhu cầu:

Index of /pub/nasm/releasebuilds/2.16.03/win64				
Name	Last modified	Size	Description	
 Parent Directory		-		
 nasm-2.16.03-installer-x64.exe	2024-04-17 10:08	1.0M	Installable package	
 nasm-2.16.03-win64.zip	2024-04-17 10:08	502K	Executable only	

Hình 40: Tải và cài đặt NASM (2)

Cài đặt NASM với quyền Administrator để đảm bảo mọi thứ hoạt động đúng yêu cầu:



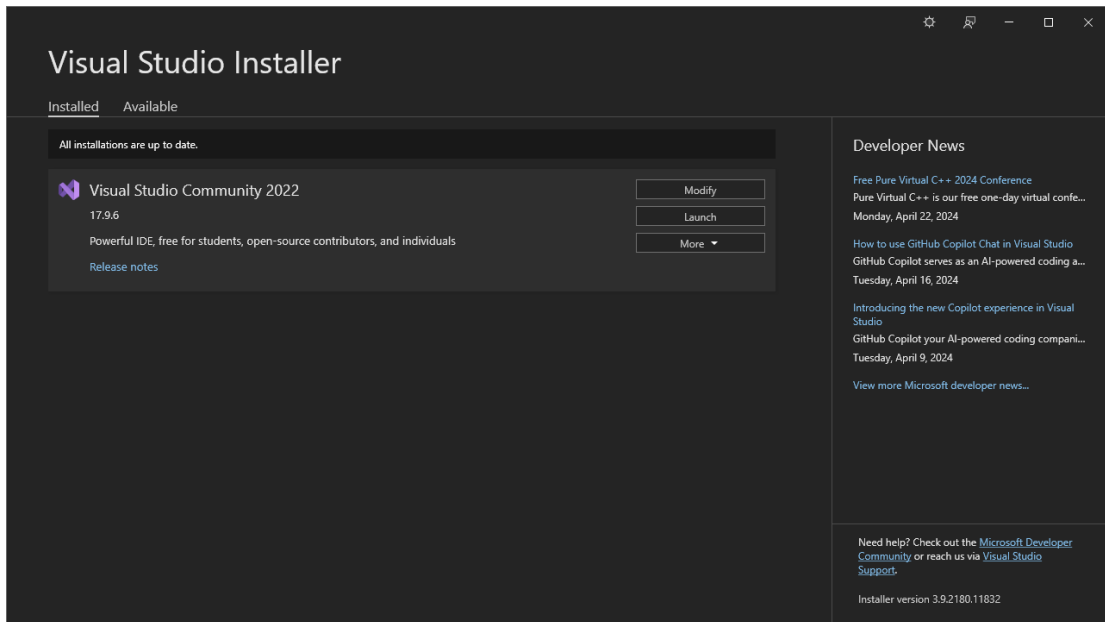
Hình 41: Tải và cài đặt NASM (3)

Sau khi cài đặt có thể sẽ cần thiết phải thêm NASM vào [biến môi trường](#)^[tr.59] toàn cục để Windows nhận bản NASM mới nhất, và sử dụng trong toàn cục.

3. Cài đặt gói phát triển với C++ của Visual Studio

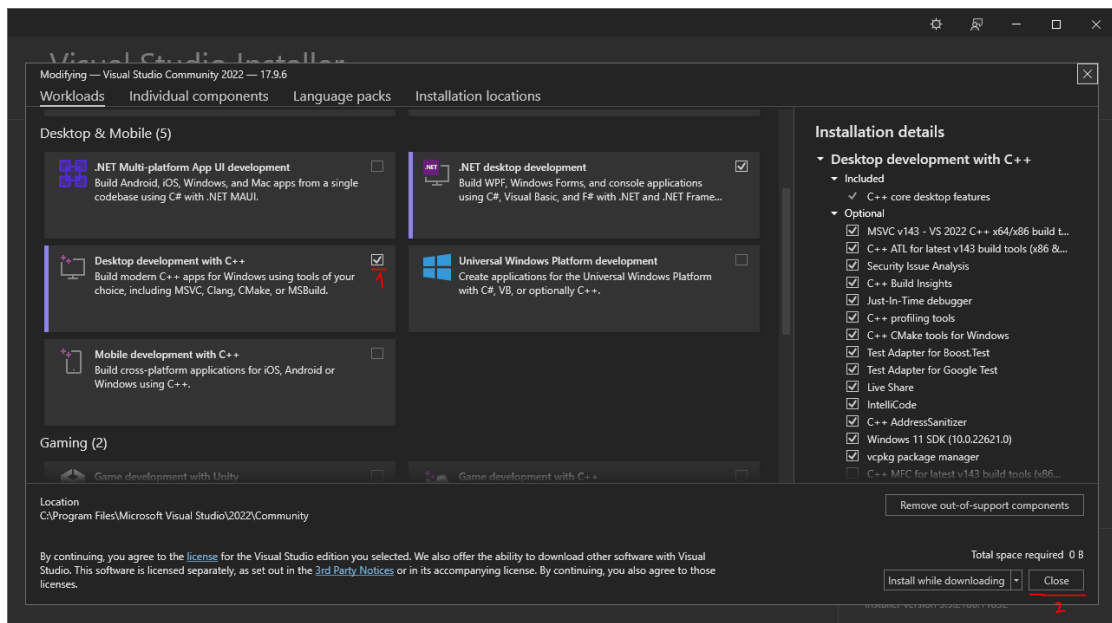
Trước tiên, tải công cụ hỗ trợ cài đặt Visual Studio (Visual Studio Installer) từ trang chính thức.

<https://visualstudio.microsoft.com/vs/>



Hình 42: Cài đặt gói phát triển với C++ của Visual Studio (1)

Tại đây có thể chọn cài đặt Visual Studio Community 2022 cùng các gói phát triển đi kèm, mình đã cài Visual Studio Community 2022 từ trước, nên mình sẽ bỏ qua bước cài, và chọn **Modify** để cài thêm các gói cần thiết cho Visual Studio Community, ở trường hợp này là gói phát triển với C++:



Hình 43: Cài đặt gói phát triển với C++ của Visual Studio (2)

Tại đây thực hiện các bước tương ứng, chọn “Desktop development with C++”(1) sau đó chọn **Install** (2) (Ở đây mình đã cài sẵn nên không hiện **Install**)

Sau khi cài, có thể tìm thấy các công cụ của C++ đã được cài đặt:

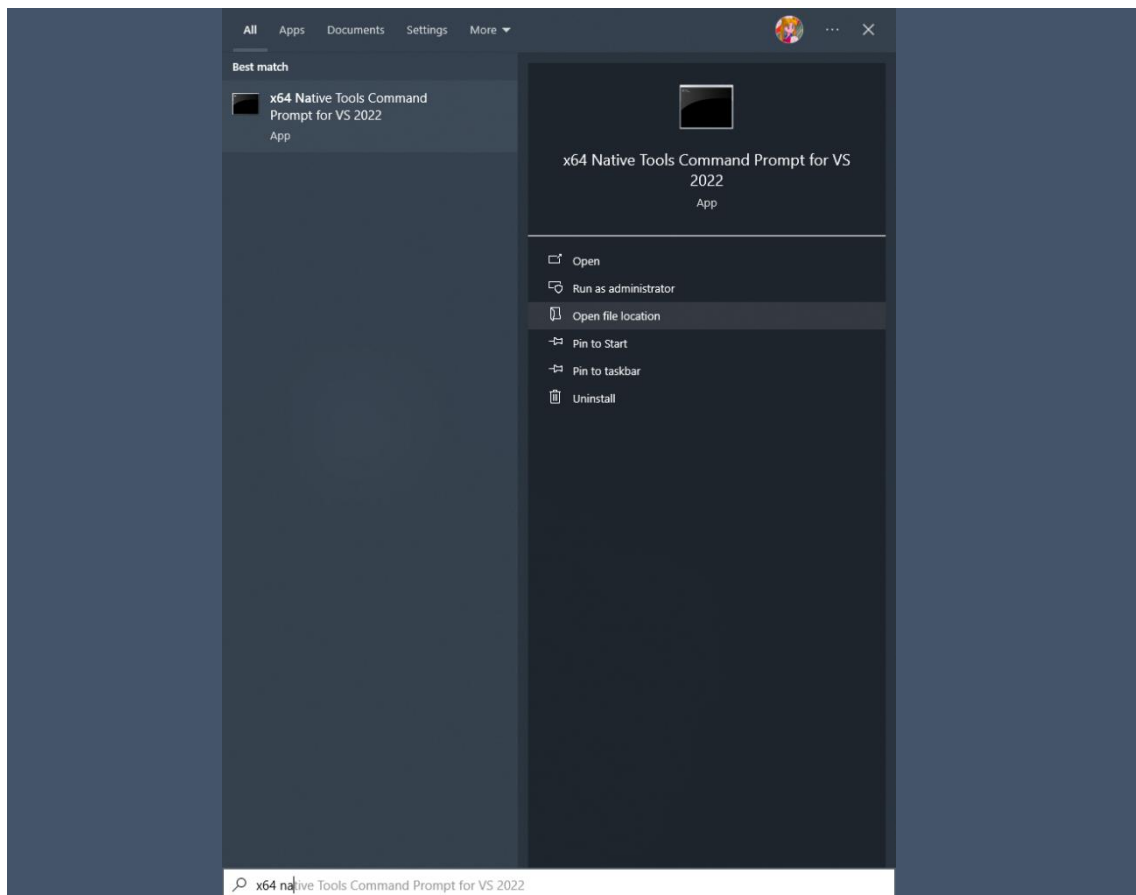
C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Tools\MSVC\14.xx\bin\Hostx64\x64

Trong đó 14.xx là phiên bản của Visual Studio, và **nmake.exe** là chương trình mình sẽ dùng.

Ngoài ra, chúng mình cũng cần môi trường phát triển **x64 Native Tools Command Prompt for VS 2022**, cũng là một phần cài đặt của gói này.

Để kiểm tra vị trí của **x64 Native Tools Command Prompt for VS 2022** hoặc để mở nó, có thể tìm theo đường dẫn hoặc tìm kiếm bằng công cụ search:

C:\ProgramData\Microsoft\Windows\Start Menu\Programs\Visual Studio 2022\Visual Studio Tools\VC



x64 Native Tools Command Prompt for VS 2022

4. MSYS2



Hình 44: logo MSYS2

4.1. Giới thiệu

MSYS2 là một môi trường phát triển tích hợp cho Windows, cung cấp các công cụ GNU và Unix dành cho việc xây dựng và chạy ứng dụng trên nền Windows. Dưới đây là một số thông tin cơ bản về **MSYS2**:

4.2. Mục đích và tính năng

Môi trường phát triển: **MSYS2** cung cấp một môi trường phát triển hoàn chỉnh với các công cụ cần thiết như trình biên dịch (**GCC**, **Clang**), trình quản lý gói (**Pacman**), và các công cụ hỗ trợ khác (**Make**, **CMake**, **Autotools**).

Tương thích Unix: **MSYS2** cung cấp một bộ công cụ Unix-like trên Windows, giúp sử dụng các lệnh và công cụ quen thuộc từ môi trường Unix trên Windows.

Cài đặt gói dễ dàng: Bằng cách sử dụng trình quản lý gói **Pacman**, có thể dễ dàng cài đặt, cập nhật và quản lý các gói phần mềm từ kho lưu trữ của **MSYS2**.

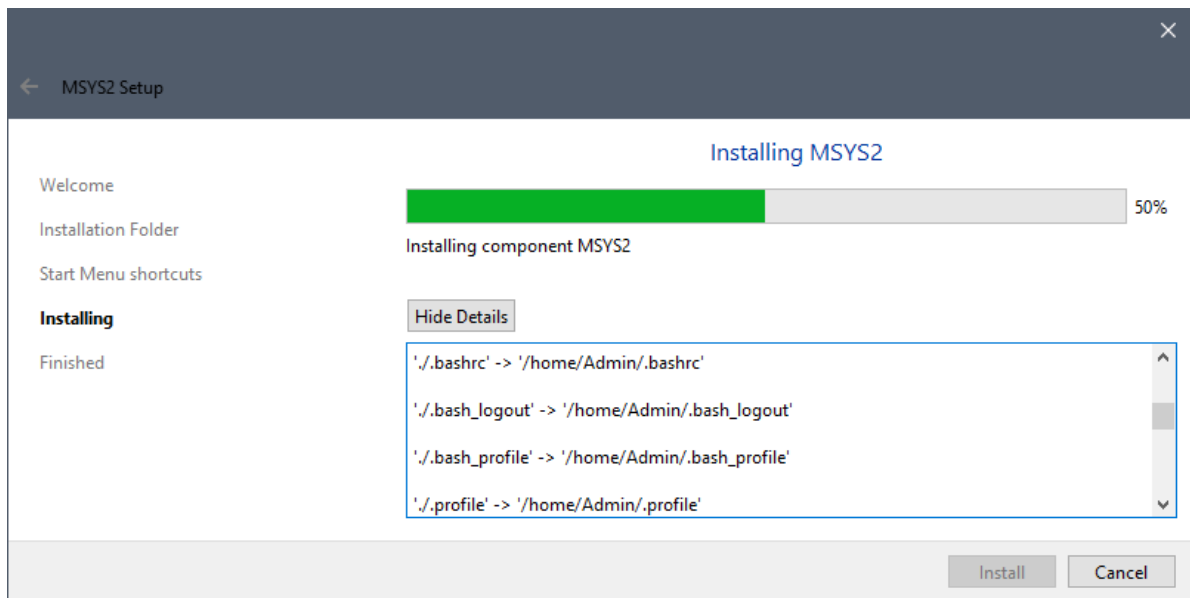
Hỗ trợ Shell: **MSYS2** đi kèm với **Bash** và một số shell khác, cung cấp một môi trường dòng lệnh mạnh mẽ trên Windows.

Cập nhật linh hoạt: Có thể dễ dàng cập nhật các công cụ và thư viện trong **MSYS2** bằng **Pacman**, giúp ta luôn có phiên bản mới nhất hoặc phiên bản mà chúng ta mong muốn.

Tóm lại: **MSYS2** là một công cụ mạnh mẽ và linh hoạt cho các nhà phát triển muốn làm việc trên nền Windows mà vẫn muốn sử dụng các công cụ và môi trường phát triển từ Unix-like.

4.3. Cài đặt và sử dụng

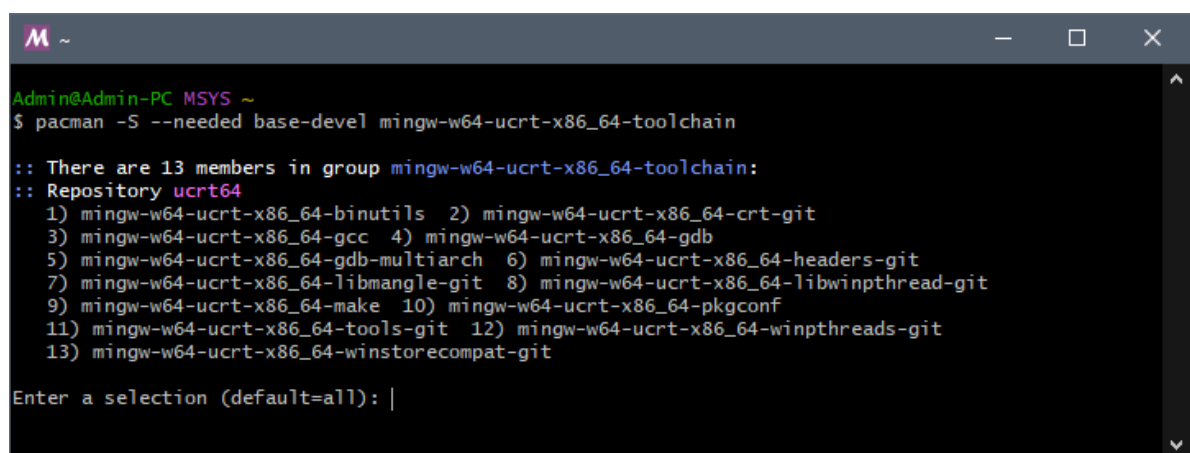
Cài đặt đơn giản: **MSYS2** cài đặt rất đơn giản bằng việc tải và chạy file cài đặt trên trang chính thức: <https://www.msys2.org/>



Hình 45: Tải và cài đặt MSYS2

Sau khi cài đặt **MSYS2**, chúng ta đã có thể sử dụng môi trường phát triển **MYS2** để cài đặt các gói thư viện hoặc compiler cần thiết cho việc phát triển. Ở đây, chúng ta cần công cụ các công cụ để **compile** và **debug** chương trình C/C++. Vậy nên chúng ta cần mở môi trường **MSYS2 MSYS** (có thể tìm bằng công cụ search)

Thực hiện lệnh sau để xem các gói thư viện phát triển có thể cài được trong gói base-level.



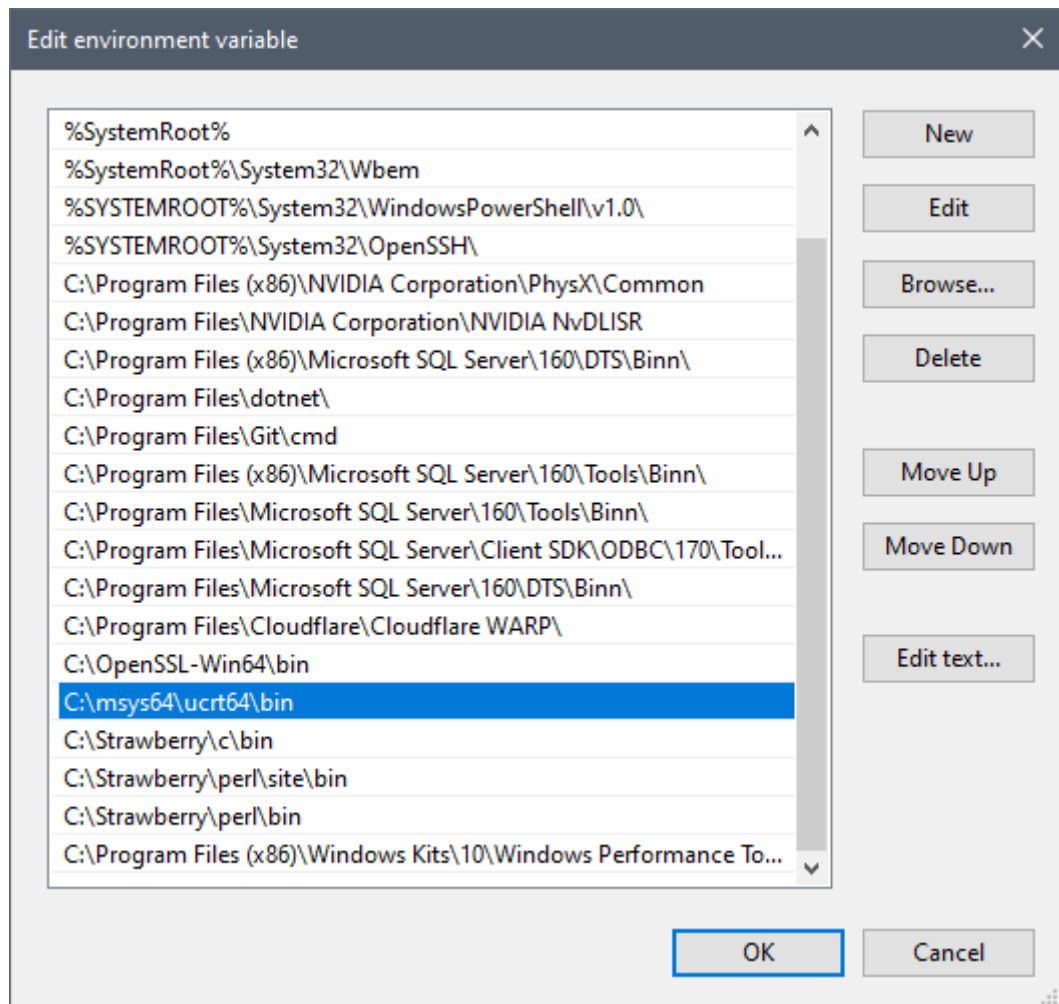
Hình 46: Tải gcc và gdb cho Windows

Ở đây, chúng ta quan tâm đến lựa chọn 3, 4 tương ứng với gói **gcc** và **gdb**. Trong gói **gcc** đã gồm cả **gcc** và **g++**.

Trên thực tế, chúng ta có thể sử dụng **gcc** trong thư viện của **Perl**. Tuy nhiên ở đây, mình muốn sử dụng cả **gcc** và **gdb** từ gói base-devel của **MSYS2** nên mình sẽ cài cả 2.

Các công cụ này để phục vụ cho việc **compile** và **debug** trong **Visual Studio Code**.

Để Windows và các ứng dụng ưu tiên sử dụng thư viện của **ucrt64 MSYS64**, thì cần thêm nó vào **PATH** và đẩy nó lên trên **Perl** như dưới đây.



Hình 47: Thêm thư viện UCRT64 vào biến môi trường

5. Windows subsystem for linux (WSL)



Hình 48: Logo WSL

5.1. Giới thiệu

Giới thiệu Windows Subsystem for Linux

WSL (Windows Subsystem for Linux) là một tính năng có trên **Windows x64** (từ Windows 10, bản 1607 và trên Windows Server 2019), nó cho phép chạy hệ điều hành Linux (GNU/Linux) trên Windows. Với WSL bạn có thể chạy các lệnh, các ứng dụng trực tiếp từ dòng lệnh Windows mà không phải bận tâm về việc tạo / quản lý máy ảo như trước đây. Cụ thể, một số lưu ý mà Microsoft liệt kê có thể làm với WSL:

- Chọn sử dụng distro Linux từ Microsoft Store: Hiện giờ đang có các **Distro Linux** rất gọn nhẹ trên Store sử dụng được ngày như **Ubuntu**, **Debian** ...
- Chạy được từ dòng lệnh các lệnh linux như ls, grep, sed ... hoặc bất kỳ chương trình nhị phân 64 bit (ELF-64) nào của **Linux**
- Chạy được các công cụ như: vim, emacs ...; các ngôn ngữ lập trình như NodeJS, JavaScript, C/C++, C# ..., các dịch vụ như MySQL, Apache, lighthttpd ...
- Có thể thực hiện cài đặt các gói từ trình quản lý gói của Distro đó (như lệnh apt trên Ubuntu)
- Từ **Windows** có thể chạy các ứng dụng **Linux** (dòng lệnh)
- Từ **Linux** có thể gọi ứng dụng của **Windows**

5.2. Cài đặt

Bước 1: Kiểm tra phiên bản Windows có tương thích hay không:

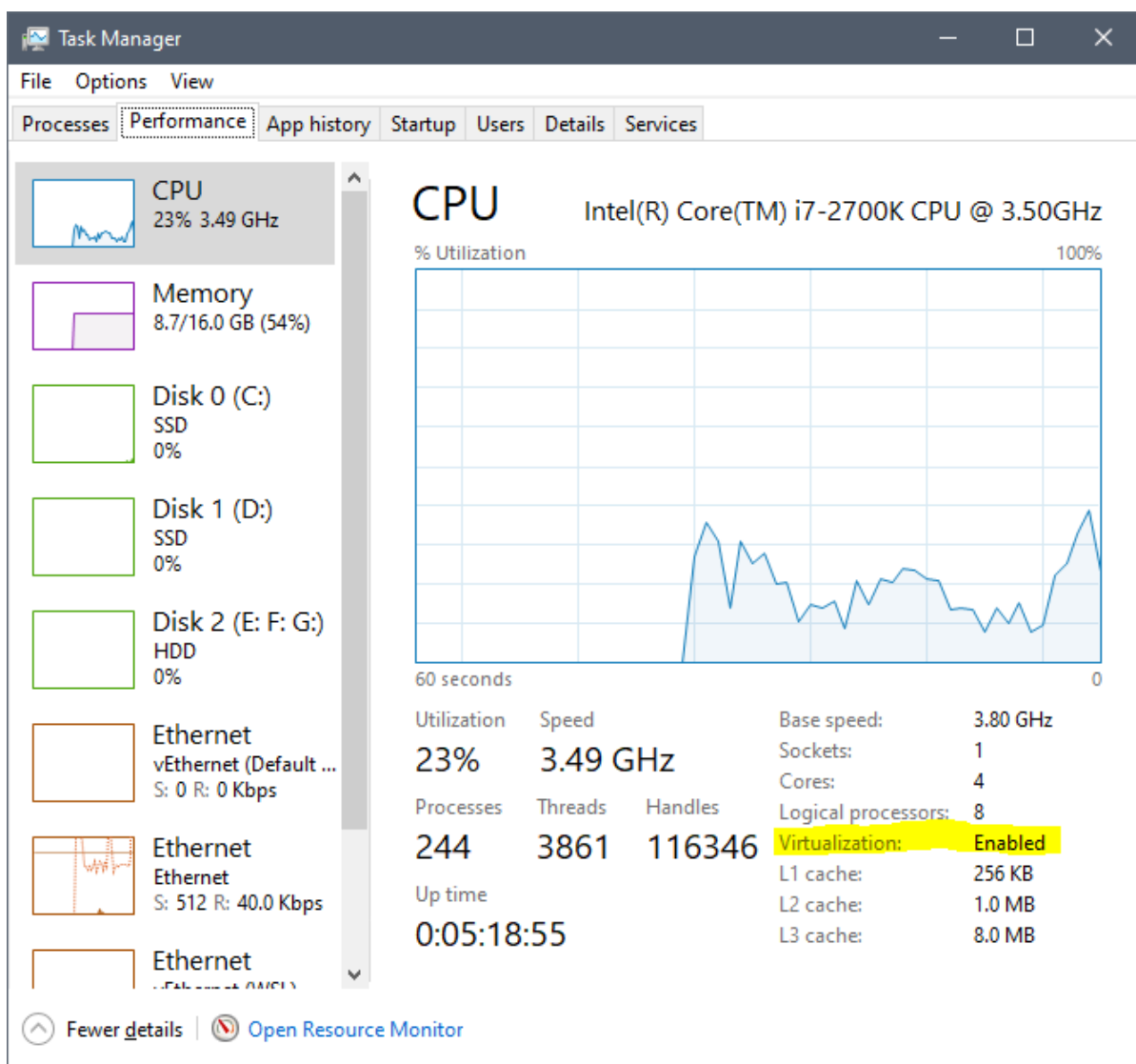
Mở **Terminal** và chạy lệnh: winver.exe



Hình 49: Kiểm tra phiên bản Windows

Ở đây **OS Build** là 1904, là cao hơn 1607 nên chắc chắn có hỗ trợ **WSL**

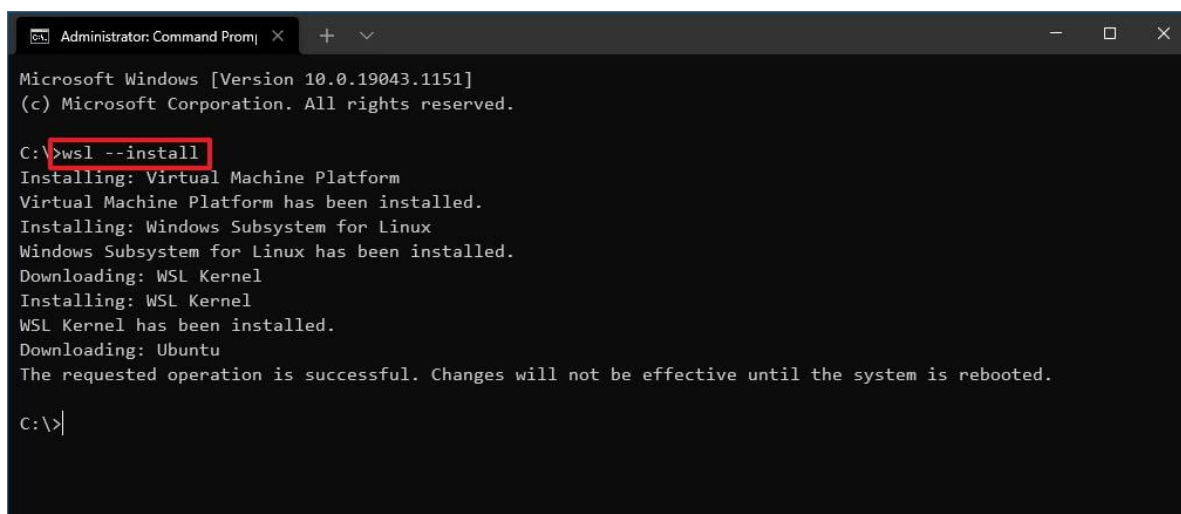
Bước 2: Kiểm tra xem chế độ ảo hoá của CPU đã được bật hay chưa (**CPU Virtualization**) dùng taskmgr.exe



Hình 50: Kiểm tra cấu hình ảo hoá CPU (Virtualization)

Đảm bảo rằng **Virtualization** đã được **enabled**, nếu là **disabled**, sẽ cần mở **BIOS** của máy tính để bật nó lên.

Bước 3: Cài đặt WSL: `wsl --install`



```
Administrator: Command Promp
Microsoft Windows [Version 10.0.19043.1151]
(c) Microsoft Corporation. All rights reserved.

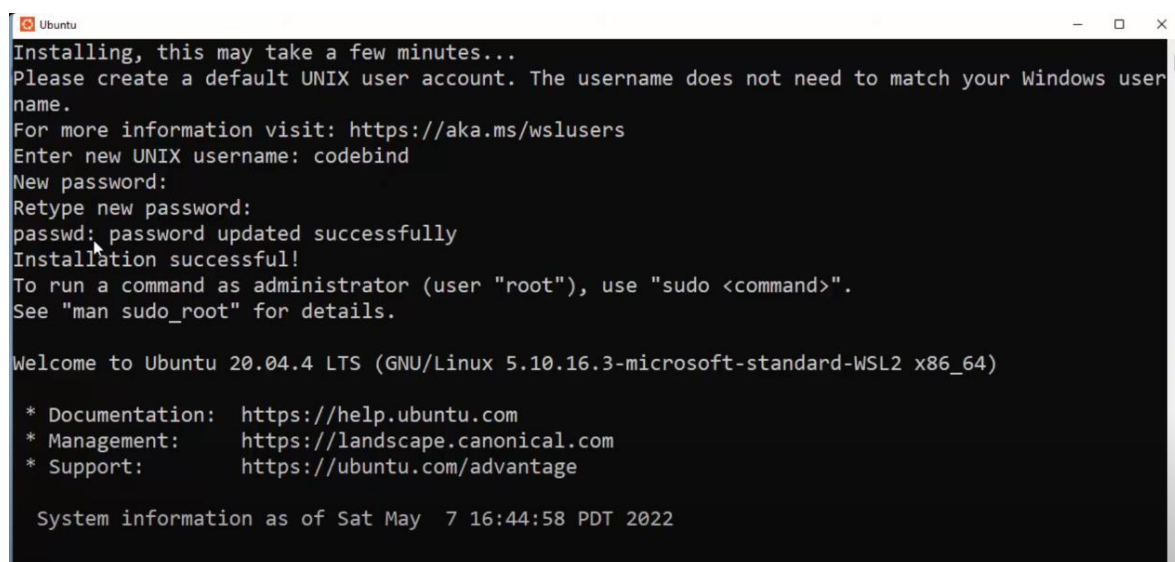
C:\>wsl --install
Installing: Virtual Machine Platform
Virtual Machine Platform has been installed.
Installing: Windows Subsystem for Linux
Windows Subsystem for Linux has been installed.
Downloading: WSL Kernel
Installing: WSL Kernel
WSL Kernel has been installed.
Downloading: Ubuntu
The requested operation is successful. Changes will not be effective until the system is rebooted.

C:\>
```

Hình 51: Cài đặt WSL

Bước 4: Sau khi cài đặt xong, mặc định chúng ta đã cài distro **Ubuntu**, chúng ta sẽ khởi động lại máy để **WSL** hoạt động.

Bước 5: Ở lần cấu hình đầu tiên khi mở bất kì **distro** nào, chúng ta sẽ cần thời gian cho distro cài đặt nốt các phần còn lại và update, sau đó **distro** sẽ yêu cầu chúng ta thiết lập tên username và password:



```
Ubuntu
Installing, this may take a few minutes...
Please create a default UNIX user account. The username does not need to match your Windows user name.
For more information visit: https://aka.ms/wslusers
Enter new UNIX username: codebind
New password:
Retype new password:
passwd: password updated successfully
Installation successful!
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

Welcome to Ubuntu 20.04.4 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:        https://ubuntu.com/advantage

System information as of Sat May  7 16:44:58 PDT 2022
```

Hình 52: Cấu hình username và password cho distro

Sau đó chúng ta đã có thể tương tác với **WSL** qua **Ubuntu**

5. Kiểm tra biến môi trường

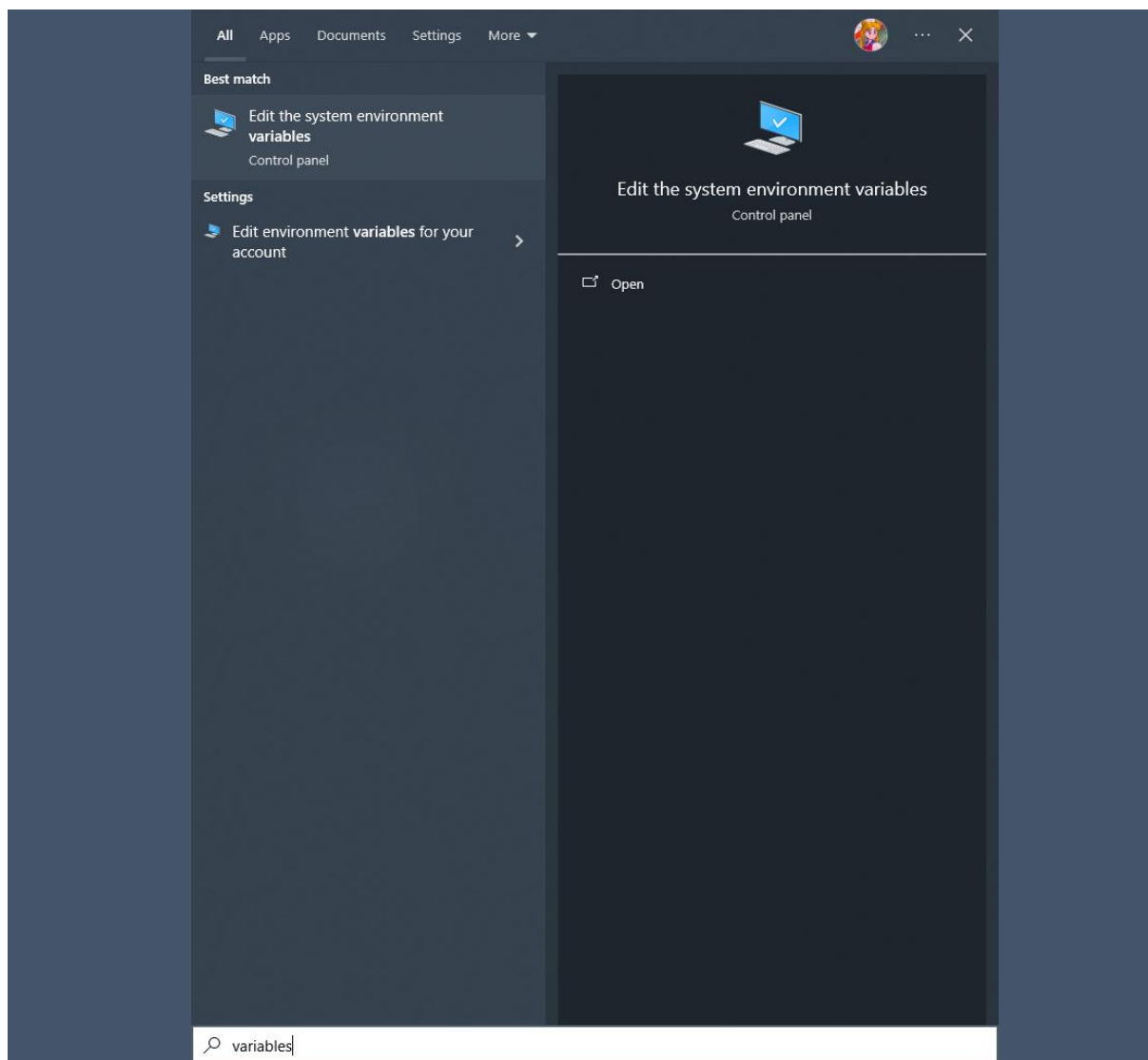
Cách 1: Bằng Terminal hoặc PowerShell hoặc Windows + R

Chạy lệnh : `systempropertiesadvanced`



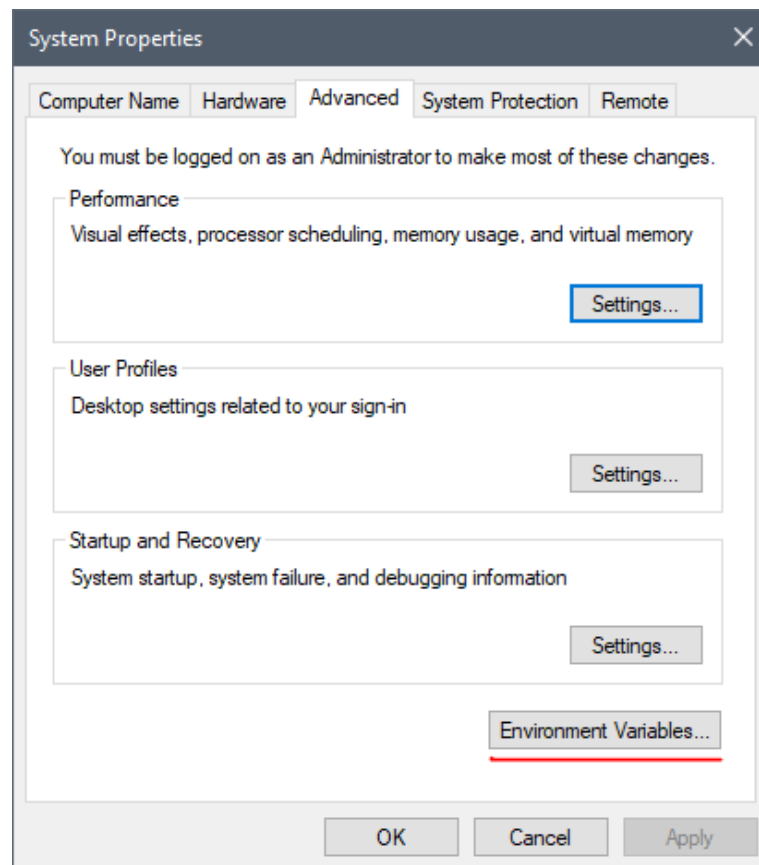
Hình 53: Kiểm tra và thêm biến môi trường (1)

Cách 2: Tìm kiếm bằng công cụ search của window



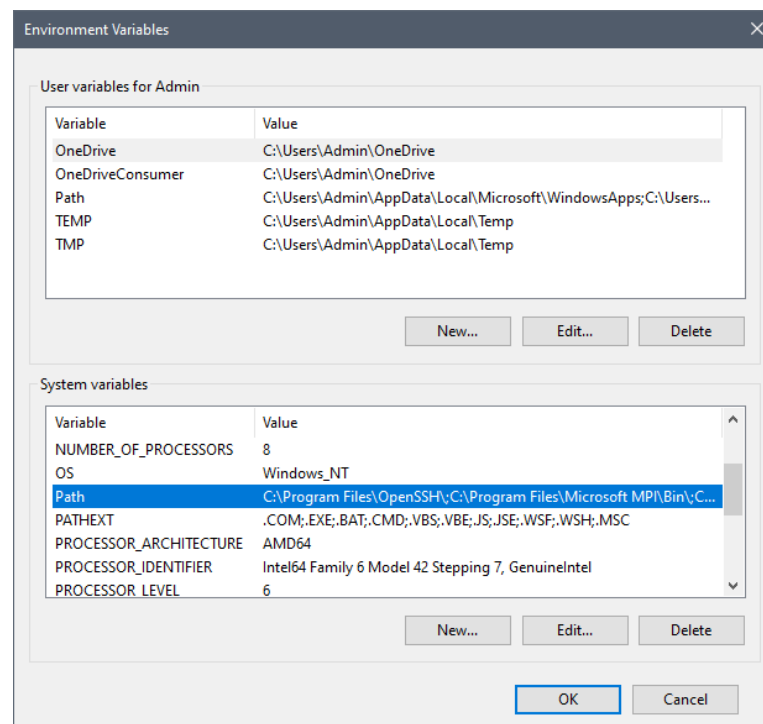
Hình 54: Kiểm tra và thêm biến môi trường (2)

Tại đây, hiện ra hộp thoại **System Properties**, quan tâm đến **Environment Variables**



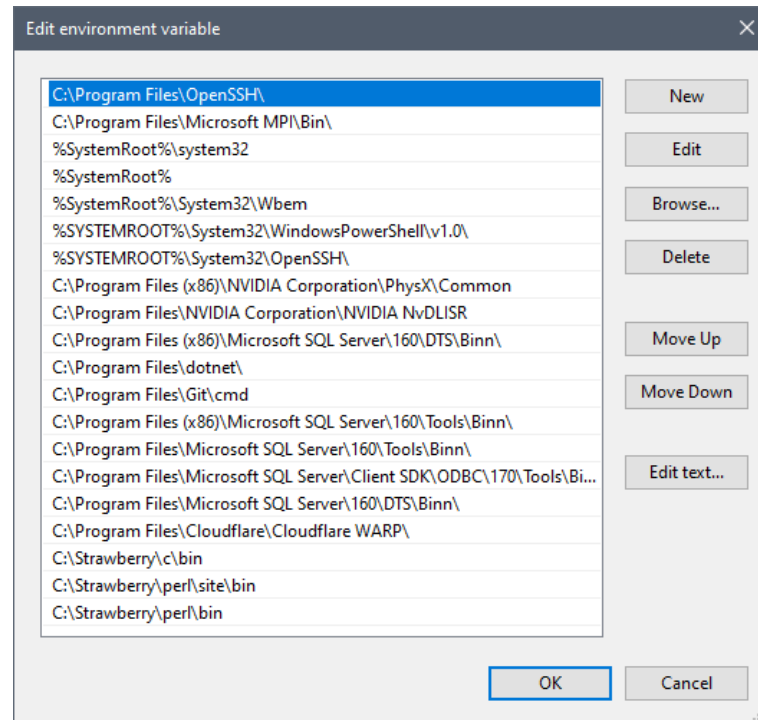
Hình 55: Kiểm tra và thêm biến môi trường (3)

Quan tâm đến đường dẫn của biến hệ thống (**System Variables**) (**Path**):



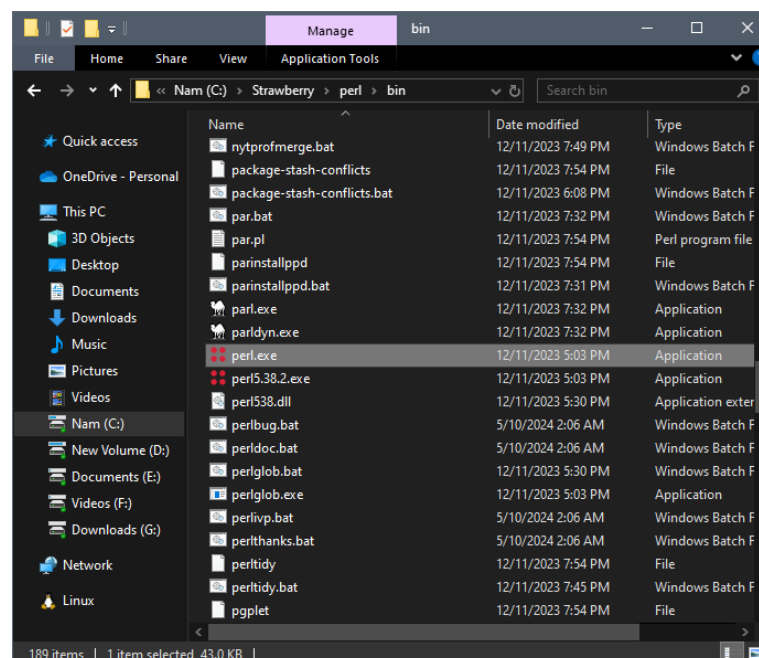
Hình 56: Kiểm tra và thêm biến môi trường (4)

Kiểm tra xem đường dẫn đến biến đã được thêm vào **Path** chưa, nếu chưa có thể thêm mới vào:



Hình 57: Kiểm tra và thêm biến môi trường (5)

Ví dụ về thêm mới:



Hình 58: Kiểm tra và thêm biến môi trường (6)

Tại folder như đường dẫn ở trên, có thể tìm thấy **Perl.exe**, chính là biến chúng ta cần, có thể thực hiện thêm folder này vào **Path**. Tương tự với những biến khác, hoàn toàn có thể thực hiện tương tự.

KẾT LUẬN

Trong quá trình thực hiện bài báo cáo này, chúng em đã trải qua một hành trình học tập và ứng dụng kiến thức đầy thú vị. Dự án không chỉ mang lại kiến thức sâu sắc về mã hoá khoá đối xứng **AES** và **OpenSSL** mà còn cung cấp cơ hội thực hành và áp dụng kiến thức vào một tình huống thực tế.

Dự án này không chỉ giúp chúng em làm chủ kiến thức chuyên môn mà còn cung cấp trải nghiệm thực tế về quá trình phát triển và triển khai mã hoá và thư viện **OpenSSL**. Chúng em tin rằng kinh nghiệm từ dự án này sẽ là nền tảng cho sự phát triển và thành công trong tương lai.

Cuối cùng, chúng em hy vọng rằng dự án này không chỉ đáp ứng được yêu cầu ban đầu mà còn mang lại giá trị và hứa hẹn cho các phát triển và nâng cấp trong tương lai. Chúng em rất tự hào về những thành tựu đã đạt được và cam kết tiếp tục nỗ lực để phát triển hơn nữa.

Tài liệu tham khảo

- Microsoft. (2024). *What is the Windows Subsystem for Linux?* Được truy lục từ Microsoft: <https://learn.microsoft.com/en-us/windows/wsl/about>
- OpenSSL. (2024). *OpenSSL - Download*. Được truy lục từ OpenSSL: <https://www.openssl.org/source/>
- OpenSSL. (2024). *OpenSSL - Release*. Được truy lục từ Github: <https://github.com/openssl/openssl/releases>
- Wikipedia. (2024). *Netwide Assembler*. Được truy lục từ Wikipedia: https://en.wikipedia.org/wiki/Netwide_Assembler
- Wikipedia. (2024). *Strawberry Perl*. Được truy lục từ Wikipedia: https://en.wikipedia.org/wiki/Strawberry_Pperl#References
- Wikipedia. (không ngày tháng). *OpenSSL*. Được truy lục từ Wikipedia: <https://en.wikipedia.org/wiki/OpenSSL>

¹ **Apache License**: https://en.wikipedia.org/wiki/Apache_License

² **Trusted Third Party** (Bên thứ ba đáng tin cậy): https://csrc.nist.gov/glossary/term/trusted_third_party

³ **Certification Authority**: <https://www.nist.gov/glossary-term/19991>

⁴ **Linux Distribution (Distro)**: https://en.wikipedia.org/wiki/Linux_distribution