物聯網應用資訊安全與風險管理

金鑰管理實作與 Apache 網頁伺服器架設

組員：劉定睿、曾彥輔、羅勻瑄、黃世君

日期：2025-10-22

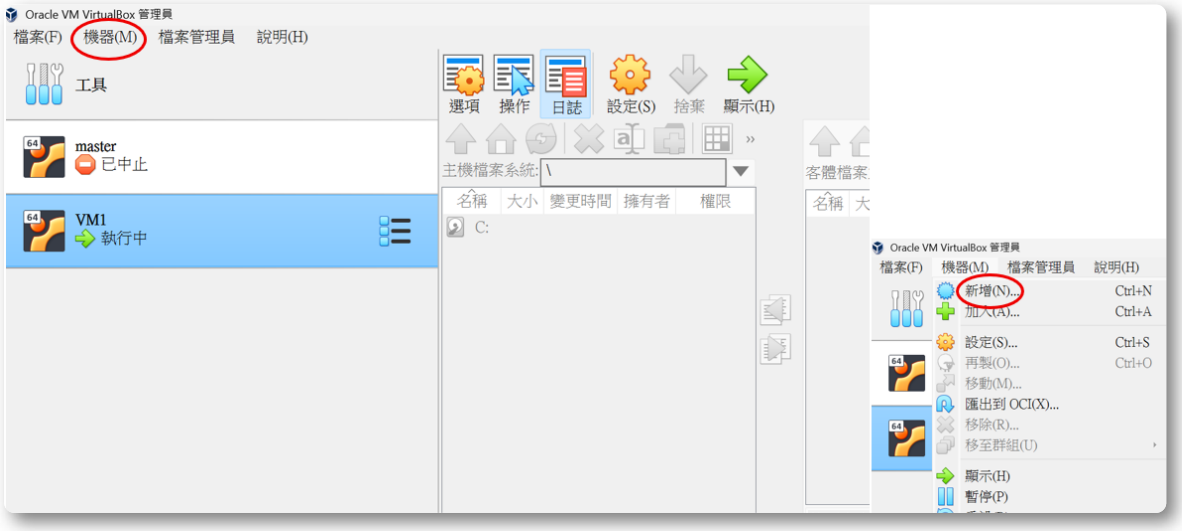
目錄

Linux 環境安裝，建立兩台機器可以網路互通的實驗環境

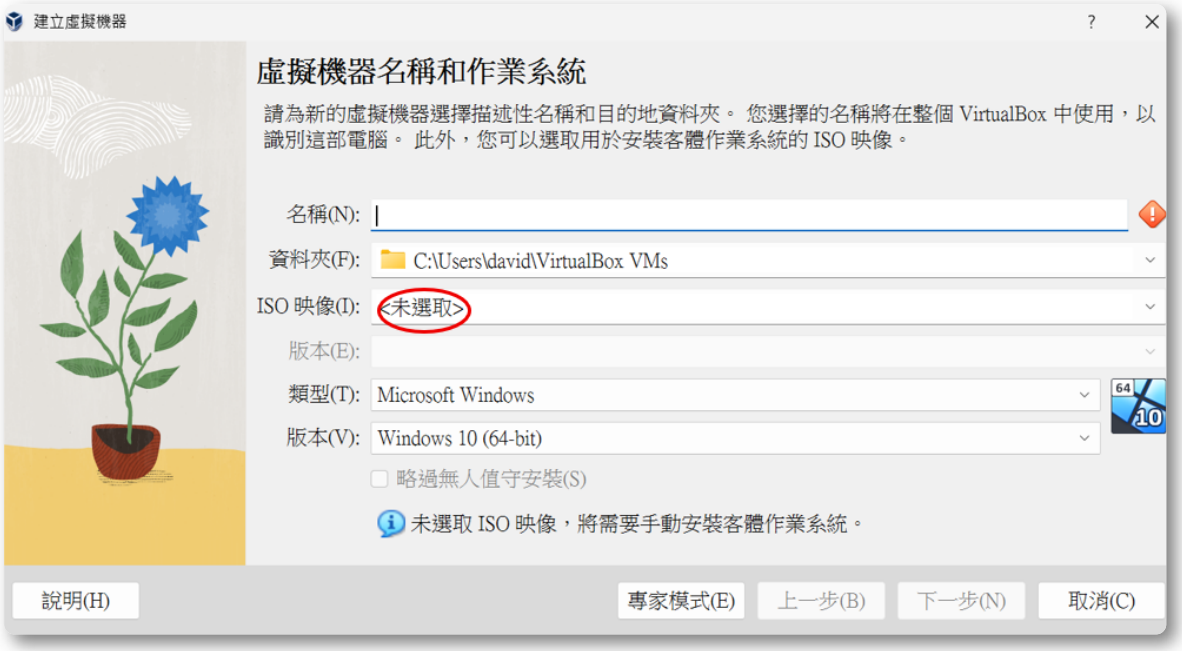
我採用的方法是在VirtualBox安裝ubuntu環境，首先要先有Oracle VM VirtualBox

1. 到https://ubuntu.com/download/server 下載映像檔

2. 進到VirtualBox介面選擇機器並選新增



3. 選取剛剛下載的映象檔並繼續

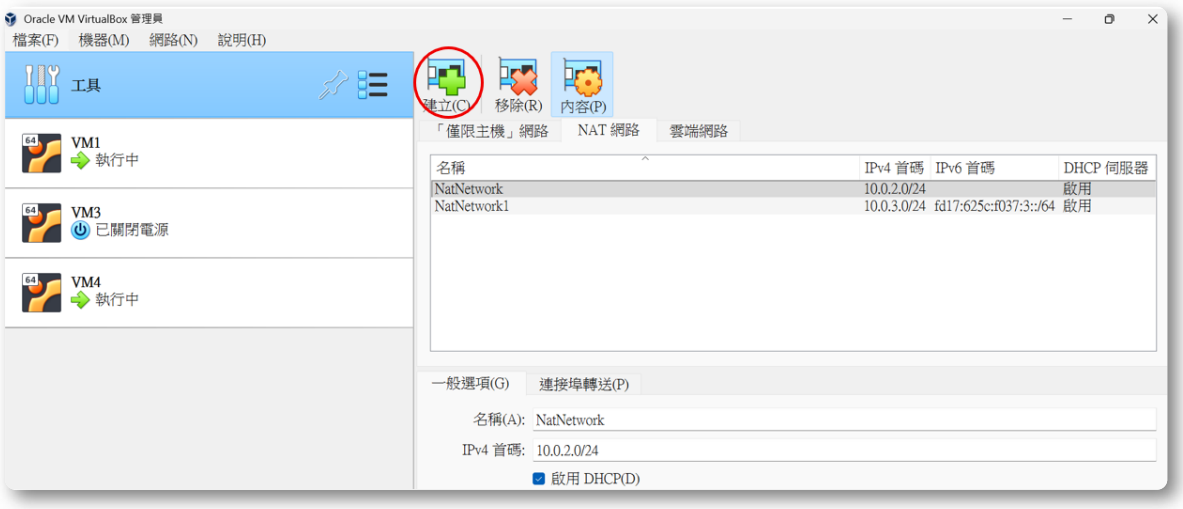


4. 在VirtualBox介面先根據需求設定磁碟分割

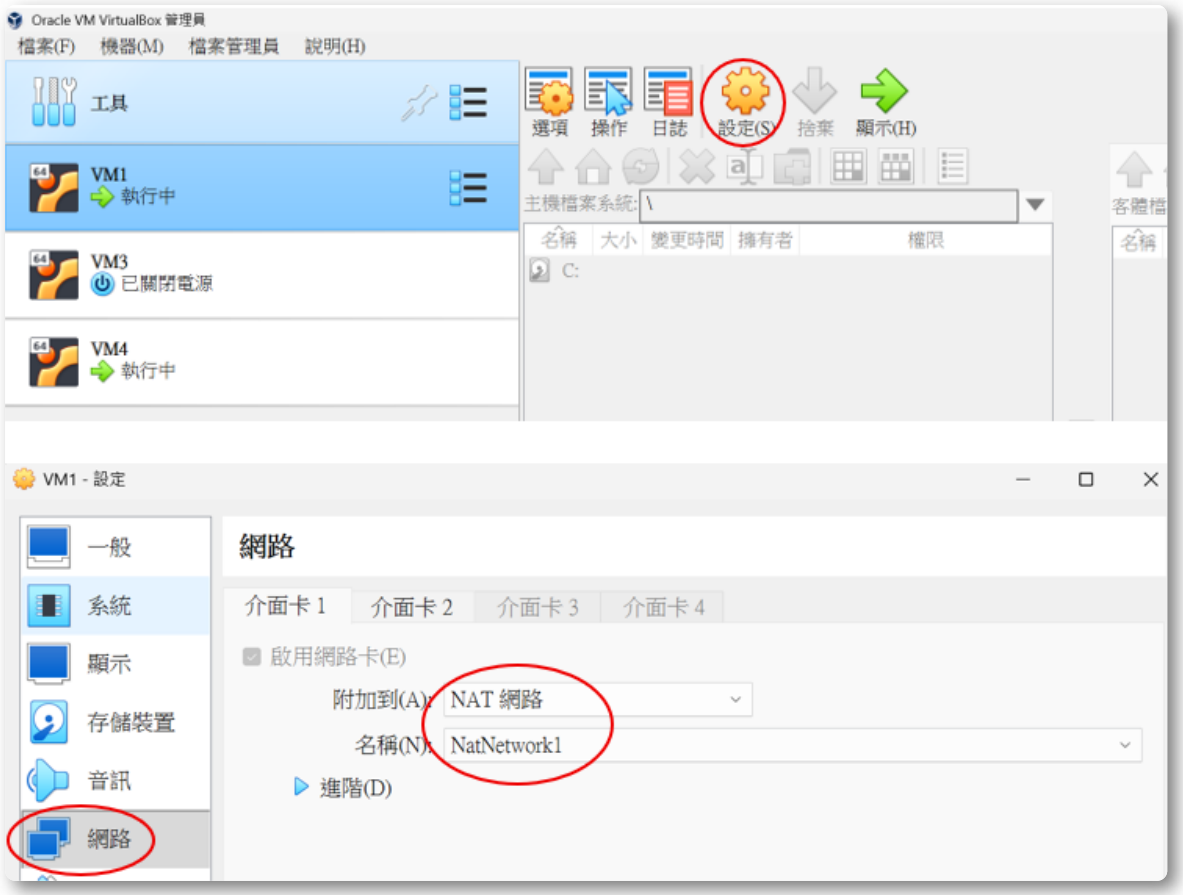
5. 開機後根據需求設定VM

6. 設定完成後先關機並重複上述步驟在安裝一台VM

7. 到工具欄位新增一個NAT network，並記得啟用DHCP

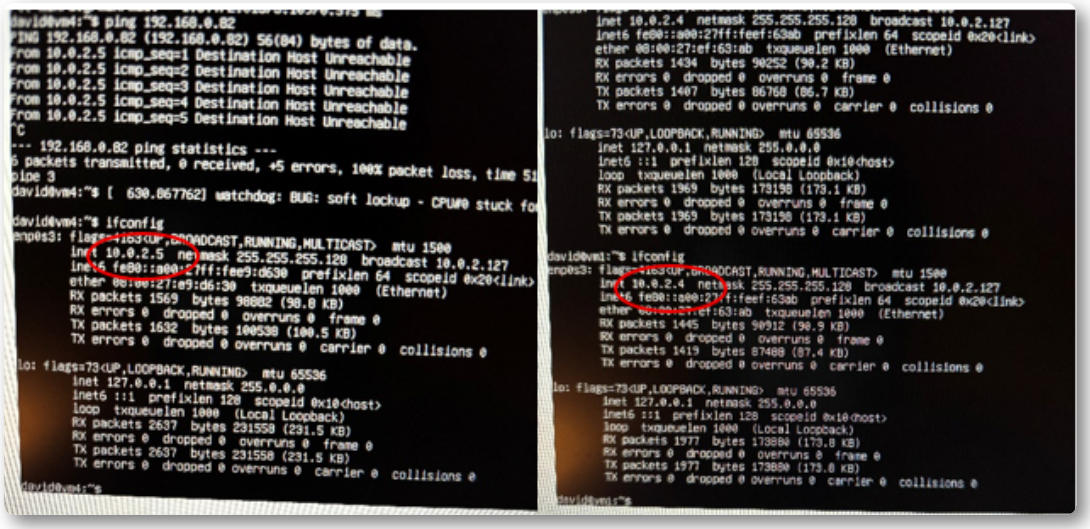


8. 兩台VM都到設定，選擇剛才新增的NAT work介面卡



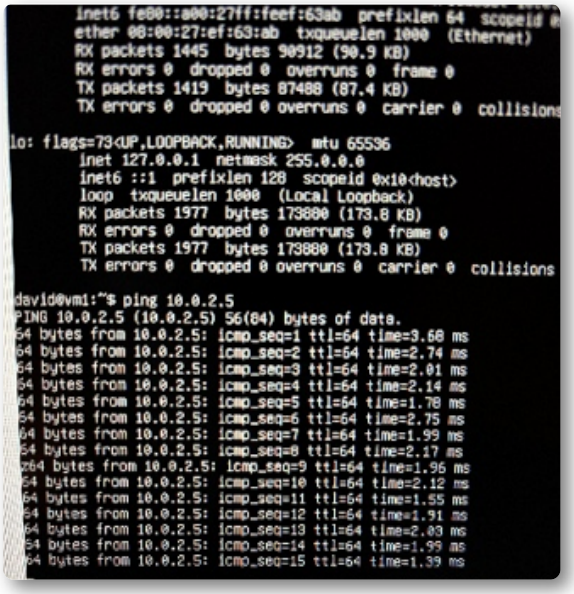
9. 設定完成後開啟兩台VM，command line輸入sudo apt install net-tools安裝網路連線套件

10. Command line輸入ifconfig查看ip位址



11. 其中一台VM去ping另一台的IP位址確認連線

    連線成功🎉🎉🎉



數位簽章實習

AES Key

- 問題背景：用 AES（對稱加密） 保護訊息機密性，並且使用 RSA + SHA-256（數位簽章） 保證傳送檔案的完整性與不可否認，並示範竄改後驗章立即失敗。

- 執行方法與步驟

1. 建立一個明文的 `secret.txt` 檔案

```
jonafk555@MSI:~$ echo 'This is the secret, 噓~' > secret.txt
jonafk555@MSI:~$ cat secret.txt
This is the secret, 噓~
```

2. 使用 openssl 建立一個 `AES Key` ，以及一個向量 `iv`

```
jonafk555@MSI:~$ openssl rand -hex 32 > aes.key.hex
jonafk555@MSI:~$ ls
2025-thu  CTFd  aes.key.hex  secret.txt
jonafk555@MSI:~$ openssl rand -hex 16 > iv.hex
jonafk555@MSI:~$ ls
2025-thu  CTFd  aes.key.hex  iv.hex  secret.txt
```

3. 將 `secret.txt` 以上面生成之 `AES Key` 、 `iv` 進行加密，並且查看加密過後密文確認已加密，輸出密文檔案 `secret.enc` 。

```
jonafk555@MSI:~$ openssl enc -aes-256-cbc -K "$(cat aes.key.hex)" -iv
"$(cat iv.hex)" -in secret.txt -out secret.enc
jonafk555@MSI:~$ cat secret.enc
WWz◆!◆ś◆_◆&MS◆a◆Ih$◆◆g◆jonafk555@MSI:~$
```

4. 接著，產生 RSA-2048 簽章金鑰對

- 產生私鑰（簽章用），輸出 `rsa.sk.pem`

```
jonafk555@MSI:~$ openssl genpkey -algorithm RSA -pkeyopt rsa_keygen_bits
:2048 -out rsa.sk.pem
.....+.........++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++*.......+....+...+.....++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++*...+........+......+...+.+.....+.+.....+.......
...+........+..........+..+.+...+............+..+..............+...+...
.++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
......++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
..............+...++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++*.+..+....+.............+.......+.....+...+...++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

- 匯出對應的公鑰（驗章用/分享用），輸出 `rsa.pk.pem`

```
jonafk555@MSI:~$ openssl rsa -in rsa.sk.pem -pubout -out rsa.pk.pem
writing RSA key
```

5. 對密文做 SHA-256(`message.enc.sig`) 數位簽章，並且確認該簽章驗證為正確。

```
jonafk555@MSI:~$ openssl dgst -sha256 -sign rsa.sk.pem -out message.enc.
sig secret.enc
jonafk555@MSI:~$ openssl dgst -sha256 -verify rsa.pk.pem -signature mess
age.enc.sig secret.enc
Verified OK
```

6. 再以 openssl 進行解密，確認輸出檔案之內容與原先明文之結果相同。

```
jonafk555@MSI:~$ openssl enc -d -aes-256-cbc -K "$(cat aes.key.hex)" -iv
 "$(cat iv.hex)" -in secret.enc -out message.dec.txt
jonafk555@MSI:~$ cat message.dec.txt
This is the secret, 噓~
```

7. 最後嘗試把原本的密文第 2 個位元組 XOR 0x01，讓密文被竄改。

```
jonafk555@MSI:~$ printf '\x01' | dd of=secret.enc bs=1 seek=1 count=1 co
nv=notrunc
1+0 records in
1+0 records out
1 byte copied, 6.1835e-05 s, 16.2 kB/s
```

8. 簽章驗證就會出現錯誤。

```
jonafk555@MSI:~$ openssl dgst -sha256 -verify rsa.pk.pem -signature mess
age.enc.sig secret.enc
Verification failure
```

- 結論：
  本次實驗實作了使用 `openssl` 生成 AES Key 與向量進行明文加密，之後再利用 `openssl` 做簽章，以確保密文不被竄改，且實作了沒竄改與有竄改的密文在 `openssl` 的 `-verify` 進行簽章驗證的結果不同。除了學習到了金鑰實作也學習到了簽章實作與流程原理。

RSA Key

- **問題背景**：
  本實驗採用 **RSA 非對稱加密** 技術，利用公鑰加密與私鑰解密來確保訊息的機密性。
  本實驗將示範加密、解密、簽章與驗章流程，並觀察訊息遭竄改後驗章失敗的情形，以了解 RSA 在資訊安全中防止竄改與否認的重要性。

- **執行方法與步驟**

1. 產生一個明文的文字檔
   指令：`vim secret.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:35:12)──> vim secret.txt                                                          ──(日,1019)─┘
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:35:36)──> cat secret.txt                                                          ──(日,1019)─┘
Don't tell your secrets to others.
```

2. 產生一把「私鑰（Private Key）」

   指令： `openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:35:39)──> openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048
.....+...+...+......++++++++++++++++++++++++++++++++++++++++*....+.......+...+...+.................+.....
..............+...++++++++++++++++++++++++++++++++++++++++*......+.....+.......+......++++++
.+.....+.....................+............++++++++++++++++++++++++++++++++++++++++*..+.........+..+....+
......+...............+..+.......+...........+....+.....+...+..........+..+....+...........+...+..+....
...+..+....+.........+........+....+.....+.+..+...........+......+...+.+........+.......+++++++++++++++
++++++++++++++++++++++*.............+..+......+.+...+...+.............+.+...+...........+..+.......
.....+.............+...+...+..+....+......+...+...........+......+....++++++
```

3. 從剛才的私鑰中導出「公鑰（Public Key）」

   指令： `openssl genpkey -algorithm RSA -out private_key.pem -pkeyopt rsa_keygen_bits:2048`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:35:55)──> openssl rsa -pubout -in private_key.pem -out public_key.pem                ──(日,1019)─┘

writing RSA key
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:36:33)──> ls                                                                      ──(日,1019)─┘
private_key.pem public_key.pem  secret.txt
```

4. 使用公鑰加密檔案

   這會生成一個加密檔案 `encypted.txt`，只有持有私鑰的人能解開。

   指令： `openssl pkeyutl -encrypt -in plaintext.txt -inkey public_key.pem -pubin -out encrypted.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:42:29)──> openssl pkeyutl -encrypt -in secret.txt -inkey public_key.pem -pubin -out encrypted.txt

┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:42:46)──> ls                                                                      ──(日,1019)─┘
encrypted.txt   private_key.pem public_key.pem  secret.txt
┌──(~/RSAtest)──────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:42:48)──> cat encrypted.txt                                                       ──(日,1019)─┘
@W@"@@@#@@`D@N0x@8@@V#@Thn@@N@ @wk@A@@@@|@@>@\@@X/T!Y7@ @lS@@@^#@[@@@
                U@@;@Fi@@@@s@@kC@5sW@Tek\#@2@@@@lj@@n@@!@@_
ţvlLN+9@$@@@t9@M@@P
@@@Yl!!@n 5l @@@@@ 1@@
                 @@ p@
                      @#@@B@@@]@@F%
```

5. 使用私鑰解密檔案

   指令： `openssl pkeyutl -decrypt -inkey private_key.pem -in encrypted.txt -out`

`decrypted.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:42:53)──> openssl pkeyutl -decrypt -inkey private_key.pem -in encrypted.txt -out decrypted.txt

┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:43:45)──> ls                                                                          ──(日,1019)─┘
decrypted.txt   encrypted.txt   private_key.pem public_key.pem  secret.txt
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:43:47)──> cat decrypted.txt                                                           ──(日,1019)─┘
Don't tell your secrets to others.
```

6. 建立訊息簽章（確保完整性與不可否認性）

   指令： `openssl dgst -sha256 -sign private_key.pem -out message.sig secret.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:51:55)──> openssl dgst -sha256 -sign private_key.pem -out message.sig secret.txt

┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:52:57)──> ls                                                                          ──(日,1019)─┘
decrypted.txt   encrypted.txt   message.sig     private_key.pem public_key.pem  secret.txt
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:53:00)──> cat message.sig                                                             ──(日,1019)─┘
]wl@F@l
          WS@k@*@@@@nO@<P @@@_@@}@@%Z@@Z@@@@x@@@V@^c@u@@m@-.@ 2@@@S@@  @@c@@Y@o@@G@z@m@I@E@@b@@@@@j,@4M
@ h@/@?Il!@@@@@@@@@@@@H@@@@xU@@K@8@U@$@?@@;@v6@53,y@@@
  @LG3@@~@}@`@@o@dO@@
@@@@J@G@;@@.k@o@@%
```

7. 驗證簽章

   若正常，輸出會顯示 `Verified OK`

   指令： `openssl dgst -sha256 -verify public_key.pem -signature message.sig secret.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:53:05)──> openssl dgst -sha256 -verify public_key.pem -signature message.sig secret.txt

Verified OK
```

8. 模擬竄改後驗章失敗

   指令： `vim secret.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:54:08)──> vim secret.txt                                                              ──(日,1019)─┘
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(22:58:00)──> cat secret.txt                                                              ──(日,1019)─┘
Don't tell my secrets to others.
```

   再次做驗正，看到 `Verification Failure` 證明驗證失敗。

   指令： `openssl dgst -sha256 -verify public_key.pem -signature message.sig secret.txt`

```
┌──(~/RSAtest)──────────────────────────────────────────────────────────────────(sharonlo@sharonM2:s001)─┐
└─(23:01:13)──> openssl dgst -sha256 -verify public_key.pem -signature message.sig secret.txt
Verification failure
C060B7F901000000:error:02000068:rsa routines:ossl_rsa_verify:bad signature:crypto/rsa/rsa_sign.c:442:
C060B7F901000000:error:1C880004:Provider routines:rsa_verify_directly:RSA lib:providers/implementations/
signature/rsa_sig.c:1041:
```

- **結論**：
  透過本實驗可觀察到，RSA 加密可確保資料機密性，而結合 SHA-256 簽章可驗證內容完整性。當訊息被竄改後，驗章立即失敗，顯示 RSA 在資訊安全中可有效防止資料竄改與否認。

HTTPS Server 安裝教學

架設 HTTPS Web Server

1. 更新套件清單

```
sudo apt update
```

2. 安裝 Apache

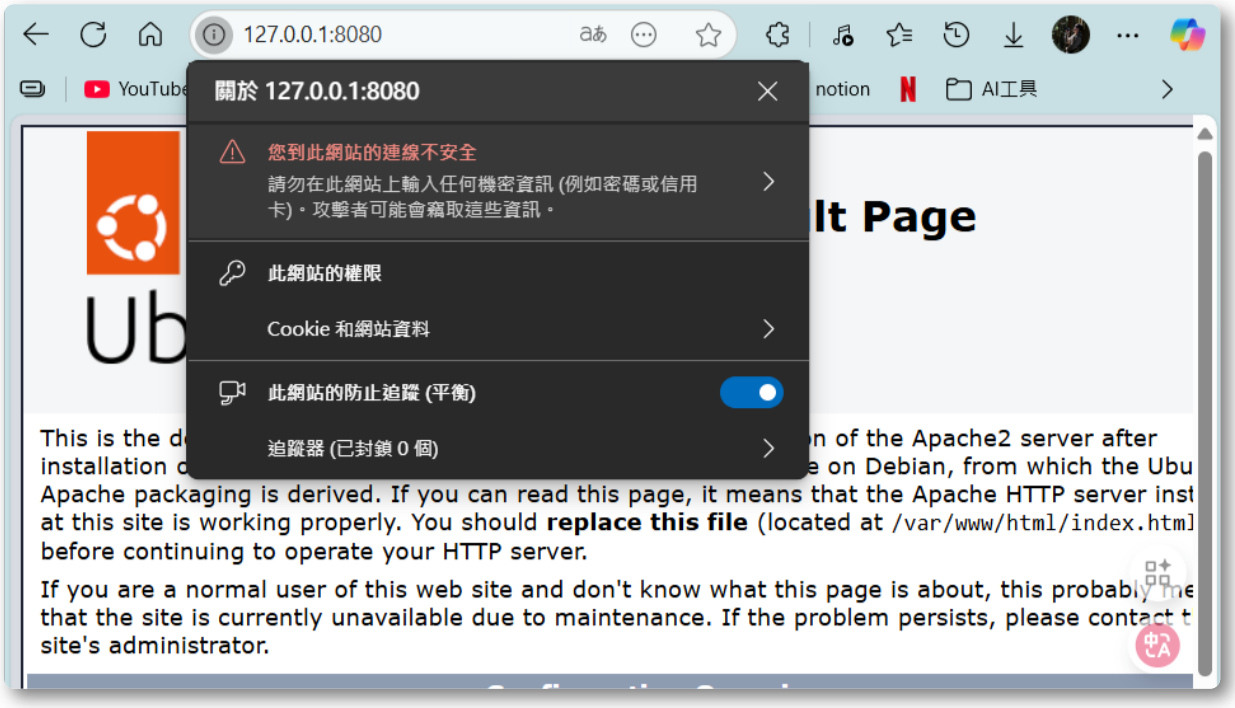
```
sudo apt install apache2
```

3. 測試 Apache

打開瀏覽器並輸入伺服器的 IP 位址：

```
http://127.0.0.1:8000
```

若一切順利，應可看到 **Apache 的預設歡迎頁面**。
此時瀏覽器會顯示「網站連線不安全」，因為目前仍使用 **HTTP** 連線。

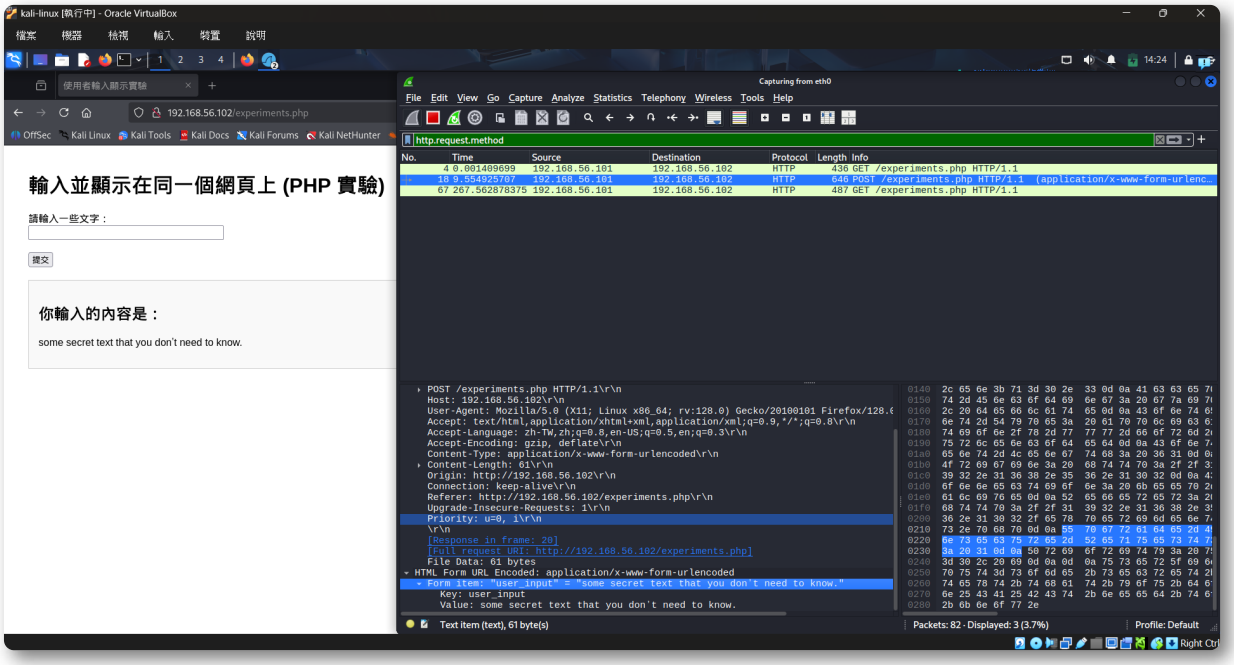


除了 Apache 本身預設的歡迎頁面外，為了更完整示範 **HTTP 與 HTTPS 的差異**，我另外製作了一個可輸入文字並顯示結果的 PHP 網頁 `experiments.php`。
在後續的實驗中，可使用 **Wireshark** 觀察輸入資訊的封包是否被監聽。

將撰寫好的 `experiments.php` 放置於：

```
/var/www/apache2
```

執行結果如下：



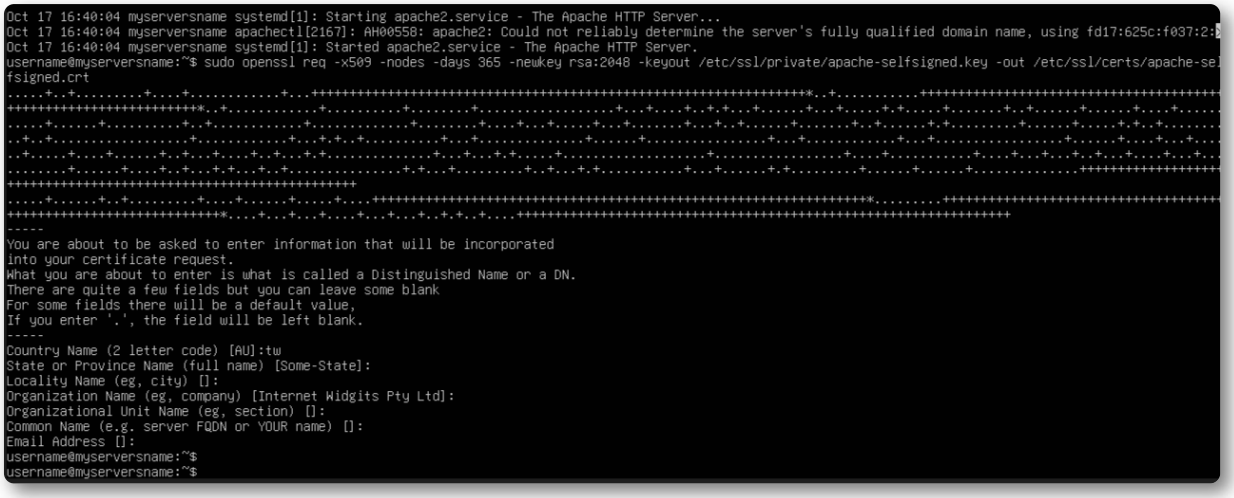
Wireshark 成功擷取到 `request.method = POST` 的封包，並清楚紀錄了我輸入的內容：

```
some secret text that you don't need to know
```

4. 生成自簽名 SSL 憑證

使用 OpenSSL 生成自簽名憑證：

```
sudo openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout /etc/ssl/private/apache-selfsigned.key \
-out /etc/ssl/certs/apache-selfsigned.crt
```



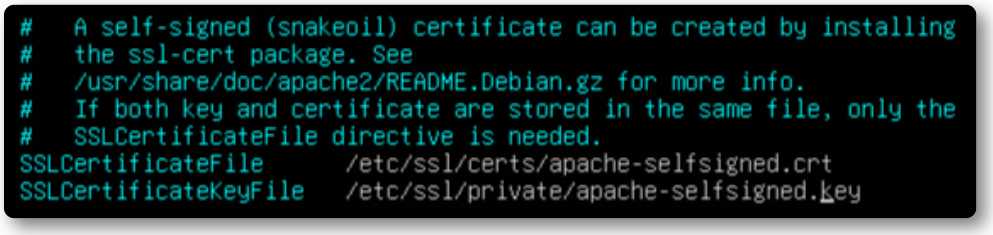
5. 設定 Apache 使用自簽名憑證

開啟 Apache 的 SSL 設定檔：

```
sudo vim /etc/apache2/sites-available/default-ssl.conf
```

找到以下兩行並修改為：

```
SSLCertificateFile /etc/ssl/certs/apache-selfsigned.crt
SSLCertificateKeyFile /etc/ssl/private/apache-selfsigned.key
```



6. 啟用 SSL 模組並啟動 HTTPS 網站

執行以下指令：

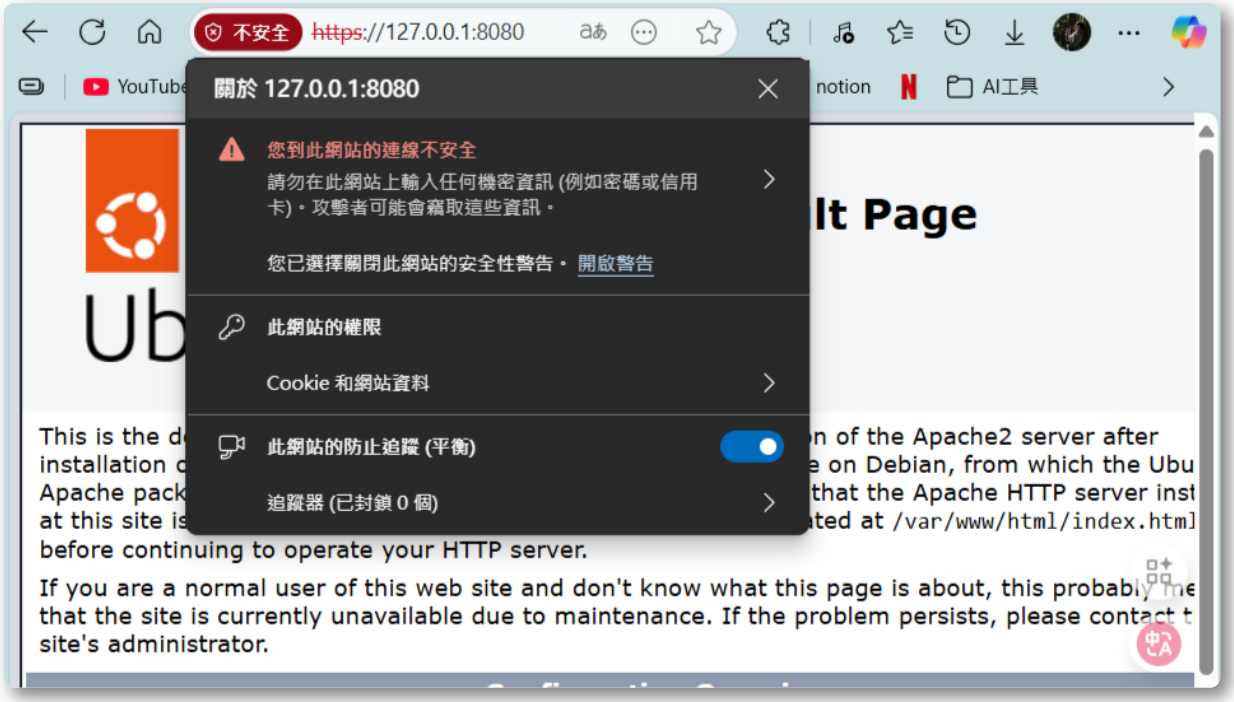
```
sudo a2enmod ssl
sudo a2ensite default-ssl
sudo systemctl restart apache2
```

7. 測試 HTTPS

在瀏覽器中輸入：

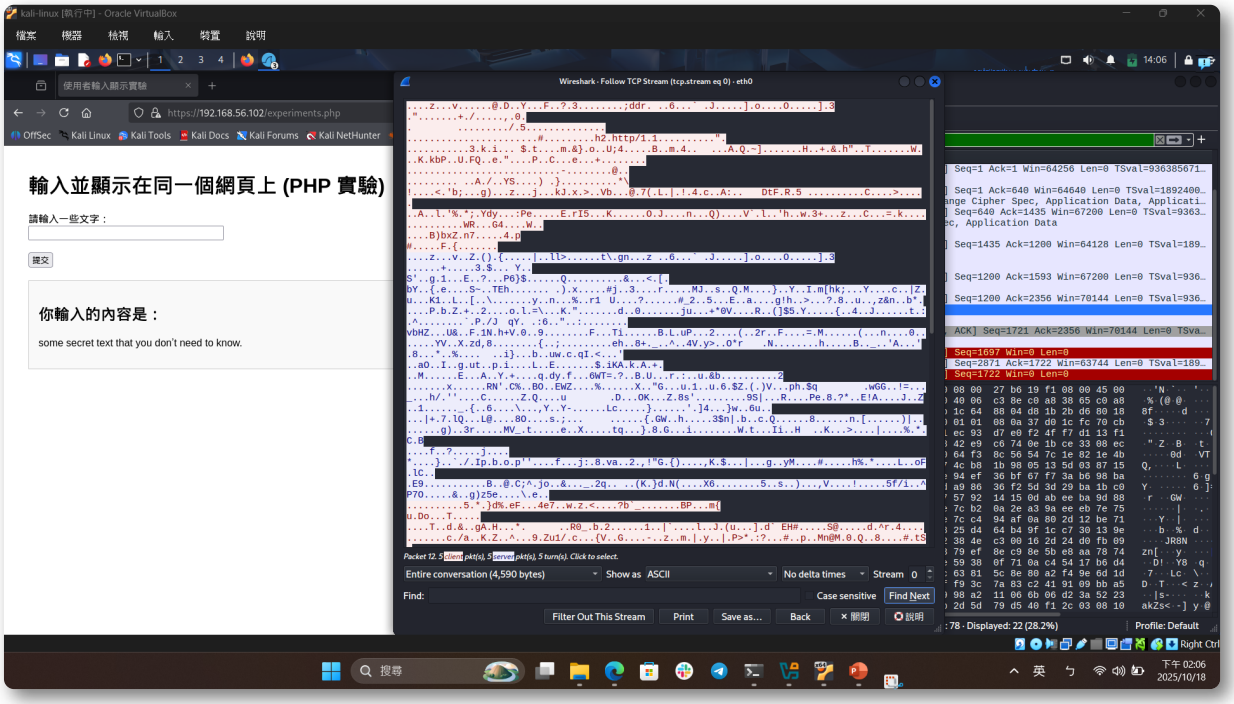
```
https://your-server-ip
```

> ⚠ 若使用虛擬機，請確認已設定 **Port Forwarding**。



此時瀏覽器可能仍顯示「連線不安全」，原因是使用了 **自簽名憑證**（未經認證機構頒發），但可觀察到連線協定已變為 **HTTPS**，資料已被加密。

接著再次測試 `experiments.php` 頁面：



可以看到 Wireshark **無法再辨識出** `request.method = POST`，也無法查看封包內容 —— 代表傳輸資料已成功被加密保護。

- 結論

  ✅ 成功完成 **HTTPS Server 架設**

  - HTTP 連線：資料可被攔截、明文顯示

- HTTPS 連線：資料被加密，封包內容無法讀取

透過此實驗可清楚理解 **SSL/TLS 加密** 在網路傳輸中的重要性。