第五章 密碼系統實習環境 - OpenSLL

5-1 OpenSSL 簡介

本書除了提供各種演算法的程式範例外，也利用 OpenSSL 系統作為實習工具。請讀者勢必將它安裝起來，研讀本書時可以隨時操作練習，甚至可利用它開發出具有安全性的應用系統。隨書光碟內包含各種演算法程式範例，以及 OpenSSL 完整套件。

OpenSSL 是一套安全性軟體開發工具（Secure Software Development Key），它包含許多密碼標準規範所製作出來的安全套件，可供開發具有安全性的應用軟體。它是一套免費的自由軟體，是由 C 語言開發而成，經由編譯後可安裝於 Linux、Windows、Unix、MAC、VMS 作業系統上，可攜性非常高。早期 OpenSSL 是由加拿大 Eric A. Yang 與 Tim J. Hudson 開發而成，目前由 OpenSSL 工作小組繼續開發與維護，並隨時將最新版本公佈於官方網站（www.openssl.org）。讀者可自行下載安裝，幾乎沒有什麼版權限制，可運用於開發於非商業或商業應用系統。

如果認為 OpenSSL 僅提供 SSL（Secure Socket Layer）庫存函數呼叫，那就太小看 OpenSSL 的功能。事實上，它包含了密碼系統的公用程式（openssl 套件）與庫存函數（crypto 套件）、以及 SSL 庫存函數（ssl 套件）等三部份。

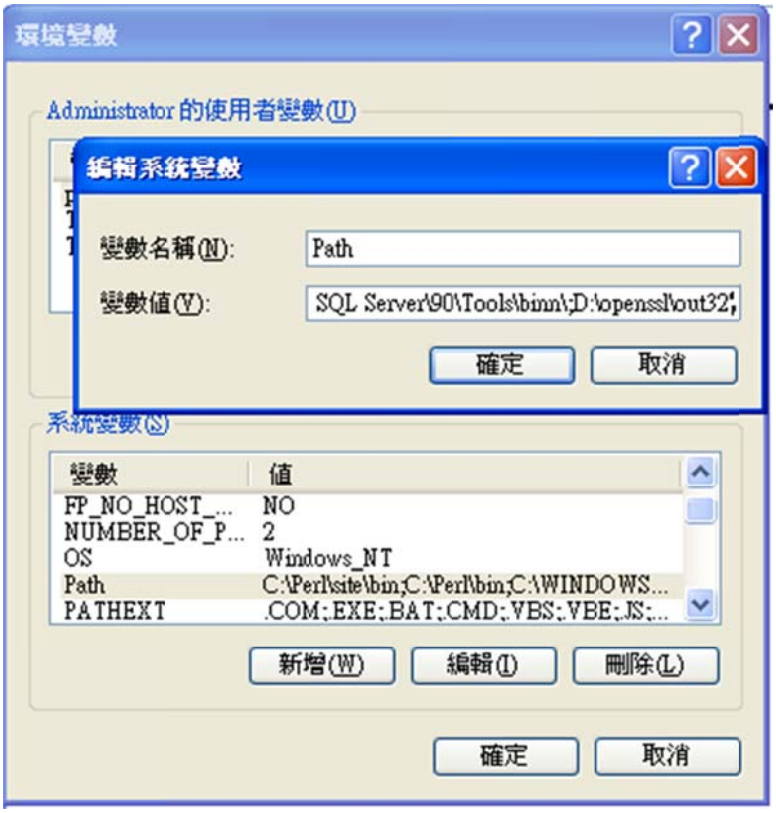
密碼系統公用程式（openssl 套件）是最可觀的『命令行』（Command line）操作工具，OpenSSL 將各種密碼演算法製作成可直接執行程式，透過這些命令操作是我們認識密碼系統的最佳途徑。它不但提供各種密碼系統的操作命令，還包含公開鑰匙產生、簽署與驗證、數位憑證產生與驗證、數位憑證發行（CA）、憑證註銷與查詢等等操作命令。透過這些公用程式，吾人可以很快速的認識密碼系統特性，與一些簡單的管理與運用。如果欲更進一步學習的話，則必須學習如何開發安全性應用系統，則非仰賴其他兩套開發工具不行。

5-2 OpenSSL 編譯與安裝

OpenSSL 最為可貴的地方，它是由 C 語言所開發出來的，任何系統只要擁有 C 編譯器，都可以將套件內所有原始程式，編譯成該系統的執行檔，再組裝成 OpenSSL 軟體發展環境。也就是說，安裝 OpenSSL 套件需要兩個步驟，首先將原始檔案編譯成可執行檔，

並建立動態連結與靜態連結之庫存函數（與安裝系統有關聯性）；第二個步驟是，利用這些庫存函數組安裝成系統發展環境（與安裝系統沒有關聯性）。老師已安裝完成並包裝成 opensll 壓縮檔，請同學複製下來解壓縮即可。

解壓縮後，必須設定 Path 變數（path=….;c:\openssl\out32\，或者安裝在 D:）。設定方法是，到系統環境視窗上設定，由桌面的『開始』→『設定』→『控制台』→『系統』→『進階』→『環境變數』→ 選 path 變數，出現視窗如圖 5-1 所示（Windows XP）。



吾人測試是否可執行，操作如下：(顯示如下，則表示安裝成功)

```
C:\openssl\OUT32>openssl
OpenSSL> ?
openssl:Error: '?' is an invalid command.

Standard commands
asn1parse         ca                ciphers           crl               crl2pkcs7
dgst              dh                dhparam           dsa               dsaparam
….
s_server          s_time            sess_id           smime             speed
spkac             verify            version           x509
```

```
Message Digest commands (see the `dgst' command for more details)
md2             md4             md5             rmd160          sha
```

5-3 OpenSSL 命令語法

5-3-1 命令格式

OpenSSL 主要操作命令是 openssl，其語法如下：

```
> openssl command [ command_opts ] [ command_args ]
```

其中 command 包含有對稱加密演算法、訊息摘要、公開鑰匙、、等等可執行的密碼系統命令；command_opts 與 command_args 是針對前項命令所需的選項與參數。執行 openssl 命令有兩種方式，一則直接執行如下：

```
H:\SecureLab\study>type data.txt
01234567890123456789012345678 9

H:\SecureLab\study>openssl sha1 data.txt
SHA1(data.txt)= 3b67544bfd2d6732fad2aabac32e692b1ae9de13
```

另一種方法是，執行 openssl 使其進入交談式運作模式，再輸入密碼系統命令，操作如下：

```
H:\SecureLab\study>openssl
OpenSSL> sha1 data.txt
SHA1(data.txt)= 3b67544bfd2d6732fad2aabac32e692b1ae9de13
OpenSSL>
```

5-3-2 通行碼輸入

執行密碼命令時，總是離不開通行碼（Password）輸入。譬如，執行簽署文件命令時，則需要輸入通行碼，方可取得私有鑰匙執行該命令；或是由私有鑰匙文件中取得公開鑰匙，也需要輸入通行碼。但該通行碼也可能運用於密碼系統的加密或解密鑰匙。由命令行輸入通行碼的途徑有：

- 提示輸入（-passin stdin）：執行命令後，系統會出現要求讀入通行碼的交談語句。
- 直接輸入（-passin pass:123456）：命令句中直接給予通行碼。
- 環境變數輸入(-passin env:passwdvar)：將通行碼儲存於某一環境變數內(passwdvar)，執行命令時再由該環境變數取出。。
- 檔案輸入（-passin file:filename）：將通行碼儲存於某一檔案內（filename），執行命令時再由該檔案內取出。

5-3-3 訊息格式

密碼系統的輸入/輸出訊息需要一套標準編碼格式，才可以在不同資訊系統之間流通，OpenSSL 採用 OSI 的 ASN.1（Abstract Syntax Notation One）標準。ASN.1 是一種結構化的數字描述語言，包含了資料描述語言（ISO 8824）與資料編碼規則（ISO 8825）兩部份。密碼套件輸入/輸出訊息大多是二進位資料檔案，或是經加密處理過的資料訊息，如果沒利用共通標準的包裝與編碼格式，可能很難流通於不同系統之間。譬如 X.509 數位憑證、數位簽章碼、訊息確認碼、、等皆是如此。

ASN.1 提供了多種資料編碼方法，包括有 BER、DER、PER 與 XER 等。這些編碼方法制定了將數字轉換成應用系統可以處理、儲存與傳輸的二進位格式，目前最普遍使用的是 BER（Basic Encode Rules）編碼規範。資料經過 ASN.1 BER 編碼後，僅將原來資料型態轉換為 BER 編碼格式，如此並無法直接流通於不同系統之間。

有了標準編碼方法（ASN.1 BER）外，吾人還需要一套標準化的訊息包裝方法，才能將電子資料（如 X.509 憑證）流通於不同系統之間。標準封裝方式除了需考慮訊息格式的共通性外，也須考慮到電子文件本身的隱密性。為了達成上述目的，吾人將流通電子文件包裝成數位信封格式，它除了具備標準封裝格式外，也具有訊息確認(MIC 功能)與隱密性(Encrypt 功能)，目前較流行的封裝規範有 PKCS #7、PGP（本書第 12 章介紹） 與 PEM 格式。

『增強隱密郵件』(Privacy Enhanced Mail, PEM，RFC 1421 ~ 1424)包含安全性郵件機制與信封封裝格式。PEM 利用資料編碼轉換與數位郵件封裝兩大步驟，來建立安全性郵件。其中，資料編碼轉換利用下列四個步驟來達成：

- 步驟 1：建立原始格式（local form）之訊息。
- 步驟 2：利用某種標準編碼方法（如 ASN.1 DER 編碼），將原始格式訊息轉換成共通文件格式。
- 步驟 3：利用對稱加密演算法（如 DES 或 AES），對已完成編碼的共通文件加密。
- 步驟 4：利用 BASE64 編碼方法（本書 12-4-2 節介紹），對已加密文件重新編碼。

吾人可將 PEM 的編碼過程表示如下：

Transmit_form = Base64 (Encrypt (DER (Loacl_form)))

接收端收到訊息後，也反向將傳輸格式轉換回原來本地格式，如下：

Local_form = de_DER (Decipher (de_Base64(Transmit_form)))

其中，de_ 表示該編碼器的反向轉換。

將訊息編碼轉換，接著需將它封裝為數位信封，大致上都採用 S/MIME 封裝技巧。本書引用 RFC 1421 範例，如下：

```
-----BEGIN PRIVACY-ENHANCED MESSAGE-----
    Proc-Type: 4,ENCRYPTED
    Content-Domain: RFC822
    DEK-Info: DES-CBC,F8143EDE5960C597
    Originator-ID-Symmetric: linn@zendia.enet.dec.com,,
    Recipient-ID-Symmetric: linn@zendia.enet.dec.com,ptf-kmc,3
    Key-Info: DES-ECB,RSA-MD2,9FD3AAD2F2691B9A,
                B70665BB9BF7CBCDA60195DB94F727D3
    Recipient-ID-Symmetric: pem-dev@tis.com,ptf-kmc,4
    Key-Info: DES-ECB,RSA-MD2,161A3F75DC82EF26,
                E2EF532C65CBCFF79F83A2658132DB47

    LLrHB0eJzyhP+/fSStdW8okeEnv47jxe7SJ/iN72ohNcUk2jHEUSoH1nvNSIWL9M
    8tEjmF/zxB+bATMtPjCUWbz8Lr9wloXIkjHUlBLpvXR0UrUzYbkNpk0agV2IzUpk
    J6UiRRGcDSvzrsoK+oNvqu6z7Xs5Xfz5rDqUcMlK1Z6720dcBWGGsDLpTpSCnpot
    dXd/H5LMDWnonNvPCwQUHt==
    -----END PRIVACY-ENHANCED MESSAGE-----
```

5-4 對稱密碼系統操作

OpenSSL 包含許多對稱密碼系統套件，不但製作成函數庫（Crypto 套件），也編寫成可執行程式（通稱為命令），讓使用者可以直接利用它完成加密或解密處理，而不用另外編寫程式。至於對稱密碼演算法相關原理與製作技巧，請參考本書第二、三章介紹。

5-4-1 命令彙集

吾人可以在 OpenSSL 交談環境裡輸入"?" 命令，觀察它提供有哪些加密演算法命令，如下：

```
H:\SecureLab\study>openssl
OpenSSL> ?
…
Cipher commands (see the `enc' command for more details)
aes-128-cbc       aes-128-ecb       aes-192-cbc       aes-192-ecb       aes-256-cbc
aes-256-ecb       base64            bf                bf-cbc            bf-cfb
bf-ecb            bf-ofb            cast              cast-cbc          cast5-cbc
cast5-cfb         cast5-ecb         cast5-ofb         des               des-cbc
des-cfb           des-ecb           des-ede           des-ede-cbc       des-ede-cfb
des-ede-ofb       des-ede3          des-ede3-cbc      des-ede3-cfb      des-ede3-ofb
des-ofb           des3              desx              idea              idea-cbc
idea-cfb          idea-ecb          idea-ofb          rc2               rc2-40-cbc
rc2-64-cbc        rc2-cbc           rc2-cfb           rc2-ecb           rc2-ofb
rc4               rc4-40
```

加密演算法大多屬於 enc 命令下的選項（演算法、加密或解密）。演算法名稱有許多共通性，吾人僅就幾種典型範例說明，即可了解大多數密碼系統，如下：

- aes-128-cbc：AES 演算法、鑰匙長度為 128 位元，採用『密文區段串接』（Cipher Block Chaining, CBC）操作模式。
- aes-128-ecb：同上，但採用『電子密碼書』（Electronic Code Book, ECB）操作。
- base64：Base64 編碼工具。

- bf：Blowfish 演算法，採用 CBC 操作模式。
- bf-cbc：同上。
- bf-cfb：同上，但採用『密文反饋』(Cipher Feedback, CFB) 操作模式。
- bf-ofb：同上，但採用『輸出反饋』(Output Feedback, OFB) 操作模式。
- cast5：Cast 5 演算法。
- des：DES 演算法。
- des-ede-cbc：兩支鑰匙的三重 DES 演算法 (E-D-E)，採用 CBC 操作模式。
- des-ede3-ofb：三支鑰匙的三重 DES 演算法，採用 OFB 操作模式。
- idea：IDEA 演算法。
- rc4：RC4 演算法。

5-4-2 命令格式 - enc

對稱加密演算法的命令格式如下：

```
openssl enc -ciphername [-in filename] [-out filename] [-pass arg] [-e]
        [-d] [-a] [-A] [-k password] [-kfile filename] [-K key] [-iv IV] [-p] [-salt]
        [-P] [-bufsize number] [-nopad] [-debug]
```

各個選項說明如下：

- -ciphername：演算法選項。
- -in filename：輸入檔案名稱。
- -out filename：輸出檔案名稱。
- -pass arg：輸入通行碼 (如 0-3-2 節介紹)。如利用於加密演算法，該選項可當加密鑰匙輸入；如運用於『私有鑰匙』加密時，則該選項視為通行碼輸入，通過確認後才可取出私有鑰匙。
- -e 或 –d：加密或解密處理，內定值是 –e (加密)。
- -K key：直接輸入加密鑰匙。
- -salt：交談式輸入醃製法中的『鹽』。如果沒有 K 與 pass 選項，則可當作加密鑰匙輸入。
- -nosalt：不採用交談式輸入『鹽』(醃製法)。

- -iv IV：操作模式的起始值（Initial Vector）。
- -nopad：明文不用填滿補期。

5-4-3 操作範例

吾人利用 DES 密碼系統做個操作範例。首先將 f1.txt 編碼加密成為 f1.bin，隨機輸入秘密鑰匙（-salt 選項，如 123456）。再將密文 f1.bin 還原存入 f1_n.txt，觀察是否與原明文相同。（在 Linux 系統下操作，Windows 下操作也相同）

```
[tsnien@Secure-1 openssl]$ cat file.txt                    【顯示明文內容】
1234567890
1234567890
1234567890
[tsnien@Secure-1 openssl]$ openssl enc -des -salt -in file.txt -out file.bin
enter des-cbc encryption password:                         【輸入秘密鑰匙：123456】
Verifying - enter des-cbc encryption password:             【秘密鑰匙：123456】
[tsnien@Secure-1 openssl]$ ls file.bin
file.bin
[tsnien@Secure-1 openssl]$ openssl enc -des -d -salt -in file.bin -out file_n.txt
enter des-cbc decryption password:                         【輸入秘密鑰匙：123456】
[tsnien@Secure-1 openssl]$ cat file_n.txt                  【顯示解密後明文】
1234567890
1234567890
1234567890
```

吾人也可以在 Windows 系統下執行相關範例，如下：

```
H:\SecureLab\study>type data.txt                           【顯示原明文內容】
01234567890123456789012345678 9
```

```
H:\SecureLab\study>openssl rc4 -in data.txt -out cipher.txt        【加密處理】

enter rc4 encryption password:                                    【輸入加密鑰匙：1234567】

Verifying - enter rc4 encryption password:                        【重複輸入加密鑰匙】


H:\SecureLab\study>openssl rc4 -d -in cipher.txt -out plain.txt    【解密處理】

enter rc4 decryption password:                                    【輸入解密鑰匙】


H:\SecureLab\study>type plain.txt                                 【顯示解密後明文】

01234567890123456789012345678 9
```

5-5 RSA 公鑰演算法操作

5-5-1 公鑰演算法命令彙集

同樣的，OpenSSL 也將許 RSA、DH 與 DSA 三種較常用的公開鑰匙系統，製作成相關命令，使用者可以直接操作。本書於第四章將會介紹到相關原理與製作技巧。吾人於 OpenSSL 交談環境中，可觀察到哪它提供有哪些命令，如下：

```
OpenSSL> ?
openssl:Error: '?' is an invalid command.

Standard commands
asn1parse       ca              ciphers         crl             crl2pkcs7
dgst            dh              dhparam         dsa             dsaparam
ec              ecparam         enc             engine          errstr
gendh           gendsa          genrsa          nseq            ocsp
passwd          pkcs12          pkcs7           pkcs8           prime
rand            req             rsa             rsautl          s_client
s_server        s_time          sess_id         smime           speed
spkac           verify          version         x509
```

以上並非所有命令皆是，吾人將公開鑰匙密碼的相關命令歸納與說明如下：

- genrsa：(RSA 演算法) 利用 RSA 演算法產生一個鑰匙檔案 (包含公開鑰匙與私有鑰匙)。
- rsa：(RSA 演算法) 處理 RSA 鑰匙的格式轉換。
- rsautl：(RSA 演算法) 使用 RSA 演算法執行加密、解密、簽署與驗證等運算。
- gendh：(DH 演算法) 產生 DH 鑰匙材料。
- daparan：(DH 演算法) 產生與處理 DH 鑰匙參數。
- dh：(DH 演算法) 處理 DH 鑰匙產生格式的轉換。
- dsaparam：(DSA 演算法) 產生與處理 DSA 鑰匙參數，利用它產生 DSA 鑰匙。
- dsa：(DSA 演算法) 處理 DSA 鑰匙的格式轉換與解譯。
- gendsa：(DSA 演算法) 依據 DSA 參數產生一個 RSA 鑰匙。

上述命令大多僅針對產生公開鑰匙，以及相關鑰匙材料，至於如何利用公開鑰匙系統簽署與驗證文件，將於 0-9 節介紹。

5-5-2 產生 RSA 鑰匙配對 – genrsa

OpenSSL 提供有 genrsa、rsa、rsautl 等三種 RSA 演算法命令。其中 genrsa 主要功能是產生 RSA 公開鑰匙與私有鑰匙配對，命令格式如下：

```
openssl genrsa [-out filename] [-passout arg] [-des] [-des3] [-idea]
        [-f4] [-3] [-rand file(s)] [-engine id] [numbits]
```

- -out filename：密鑰輸出檔案，內定值是 PEM 格式。
- -passout arg：對私有鑰匙加密的通行碼輸入。
- -des、-des3、-idea：對私有鑰匙加密的演算法。
- -f4 與 -3：公開鑰匙 e 的選項。指定 -3 則表示 e=3；如指定 –f4，則 e=65537 (內定值)。
- numbits：指定鑰匙長度，大多是 1024 或 2048。

操作範例。吾人欲產生一對 1024 位元鑰匙配對，並儲存於 rsaprivate.pem 檔案 (包含

公開與私有兩支鑰匙，一般稱為私有鑰匙檔案）內，其中私有鑰匙利用 DES 演算法加密，由鍵盤輸入通行碼作為加密鑰匙，操作如下：(Windows 系統下操作)

```
H:\SecureLab>openssl genrsa -out rsaprivate.pem -passout pass:12345 -des 1024
Loading 'screen' into random state - done
Generating RSA private key, 1024 bit long modulus
.................++++++
.................++++++
e is 65537 (0x10001)

H:\SecureLab>dir/b rsaprivate.pem
rsaprivate.pem
```

5-5-3 RSA 管理命令 - rsa

此命令是用來對已產生的鑰匙檔案管理，譬如重新設定通行碼、加密演算法、取出公開鑰匙或進行檔案格式轉換等等，命令格式如下：

```
openssl rsa [-inform fmt] [-outform fmt] [-in filename] [-passin arg]
        [-out filename] [-passout arg] [-sgckey] [-des] [-des3] [-idea]
        [-text] [-noout] [-modulus] [-check] [-pubin] [-pubout] [-engine id]
```

- -inform fmt、-outform fmt：鑰匙檔案的輸入與輸出格式，其中 fmt 編碼可選擇 d(DER)、p (PEM)、t (Text)、n (NET)、p12 (PKCS #12) 或 e (Engine) 等格式。
- -passin arg、-passout arg：前者是用於輸入通行碼，來取出私有鑰匙；後修改私有鑰匙的通行碼。
- -des、-des3、-idea：變更對私有鑰匙加密的演算法，需配合 –passout 選項輸入通行碼，否則執行螢幕會出現要求輸入通行碼語句。
- -text、-noout、-modulus：選擇鑰匙參數的輸出格式。其中 -text 則以明文格式輸出，-noout 則表示不會輸出任何鑰匙參數道輸出檔案中，-modulus 表示輸出模組格式。
- -pubin 與 –pubout：公開鑰匙輸出或輸入選項，內定值是公開鑰匙輸出 (-pubout)。

範例操作。之前吾人利用 genrsa 產生一個鑰匙檔案，接著再利用 rsa 管理命令，將他轉換成一般文件格式 (Text，rsaprivate.txt)，觀察其內容如何，操作如下：

```
H:\SecureLab\study>openssl rsa -in rsaprivate.pem -passin pass:12345 –text rsaprivate.txt
```

接著，再觀察 rsaprivate.txt 檔案內容，如下：

```
H:\SecureLab\study>type rsaprivate.txt
Private-Key: (1024 bit)
modulus:
    00:b8:e9:23:ce:62:4b:1a:d8:aa:57:2f:6b:28:83:
    a2:44:98:67:40:03:a8:2d:2e:23:99:88:a6:a8:9c:
    62:07:98:f8:9c:95:24:c6:1b:5a:07:2e:eb:49:b0:
    38:f5:28:b7:4b:a8:eb:9d:34:bb:64:96:e8:80:28:
    59:52:e7:bd:22:79:2a:66:cd:25:d9:58:4e:c9:47:
    1a:ee:b1:2c:4b:4e:80:61:b7:2c:1d:49:96:5f:b5:
    1a:07:e3:5f:47:73:14:42:b7:b7:56:d5:76:05:0d:
    3f:d9:b1:f3:62:e3:b1:1f:13:ed:8e:29:29:95:56:
    01:8b:26:f3:0b:99:2e:0a:0d
publicExponent: 65537 (0x10001)
privateExponent:
    00:ac:0d:a2:97:82:b6:47:80:9a:df:0c:ec:34:82:
    67:16:54:88:9d:f3:c9:24:60:ee:17:0a:23:a8:77:
    fa:6e:ff:53:34:bf:41:bc:63:ee:dd:08:37:3e:15:
    8e:a9:ee:fc:be:95:aa:c6:58:2e:95:66:25:68:3a:
    23:03:15:45:d1:9c:97:94:5b:31:47:3f:61:ac:20:
    49:b0:1f:15:94:3e:cf:e5:56:b0:1a:4f:49:53:76:
    99:1e:6d:24:8d:a1:a5:43:ab:9e:12:41:dd:db:d2:
    09:e6:0a:21:c1:39:50:6a:06:0a:7d:46:69:f0:cf:
    86:a5:50:85:7f:8c:17:25:b1
prime1:
    00:e2:c1:fd:c2:47:ce:84:f3:6d:ef:93:73:8f:3c:
    91:91:f9:3c:9e:4e:db:65:fc:11:80:77:14:c3:46:
    25:1d:a5:1d:04:0f:f3:37:3b:2f:7c:3e:db:06:2c:
    1d:64:de:d5:6f:1b:1f:1e:b7:7f:96:5f:11:9e:75:
    f2:54:2d:3b:7f
prime2:
    00:d0:c1:a4:1f:8c:78:ad:bc:f0:80:20:27:db:19:
    57:03:27:f7:0a:a2:94:a0:b4:11:fd:08:d3:28:f3:
    b9:cf:04:05:e8:fa:ab:e8:da:ce:ea:c4:cf:80:45:
    2a:ae:8e:29:e9:a6:e0:8d:b7:0d:f7:b0:a0:8f:ea:
    77:9e:8b:b0:73
exponent1:
    00:97:2b:b2:eb:d1:89:49:b3:2b:e8:5b:09:e0:45:
```

```
        05:db:26:28:96:75:85:e8:c0:9a:3e:65:a4:ee:e6:
        15:9a:64:d8:2a:3c:23:ed:ff:44:11:f5:a9:78:bc:
        f2:3f:ac:1a:e8:3e:51:89:dd:d5:6e:3f:24:f4:da:
        36:da:8d:69:2b
exponent2:
        00:a5:35:ba:c7:e5:09:d4:a5:c4:c8:01:aa:c9:31:
        02:b5:d2:b9:26:47:88:cc:ad:f5:d6:85:57:67:ff:
        8b:3b:94:79:80:ea:71:86:b5:34:30:84:55:9b:b4:
        21:95:47:99:4f:fa:eb:97:fc:19:27:bf:37:32:ee:
        62:80:ad:18:95
coefficient:
        51:28:5a:d1:9a:a7:60:ba:6c:94:83:44:b0:bd:d8:
        8b:9b:9f:db:05:0c:1e:d6:f5:64:97:93:46:ff:1f:
        0f:78:ea:d5:ea:4c:ba:aa:8d:27:44:68:5b:64:93:
        b7:08:f5:71:0b:3c:81:c9:7d:b7:82:8b:70:92:ef:
        47:2a:10:4d
-----BEGIN RSA PRIVATE KEY-----
MIICXgIBAAKBgQC46SPOYksa2KpXL2sog6JEmGdAA6gtLiOZiKaonGIHmPiclSTG
G1oHLutJsDj1KLdLqOudNLtkluiAKFlS570ieSpmzSXZWE7JRxrusSxLToBhtywd
SZZftRoH419HcxRCt7dW1XYFDT/ZsfNi47EfE+2OKSmVVgGLJvMLmS4KDQIDAQAB
AoGBAKwNopeCtkeAmt8M7DSCZxZUiJ3zySRg7hcKI6h3+m7/UzS/Qbxj7t0INz4V
jqnu/L6VqsZYLpVmJWg6IwMVRdGcl5RbMUc/YawgSbAfFZQ+z+VWsBpPSVN2mR5t
JI2hpUOrnhJB3dvSCeYKIcE5UGoGCn1GafDPhqVQhX+MFyWxAkEA4sH9wkfOhPNt
75NzjzyRkfk8nk7bZfwRgHcUw0YlHaUdBA/zNzsvfD7bBiwdZN7VbxsfHrd/ll8R
nnXyVC07fwJBANDBpB+MeK288IAgJ9sZVwMn9wqilKC0Ef0I0yjzuc8EBej6q+ja
zurEz4BFKq6OKemm4I23DfewoI/qd56LsHMCQQCXK7Lr0YlJsyvoWwngRQXbJiiW
dYXowJo+ZaTu5hWaZNgqPCPt/0QR9al4vPI/rBroPlGJ3dVuPyT02jbajWkrAkEA
pTW6x+UJ1KXEyAGqyTECtdK5JkeIzK311oVXZ/+LO5R5gOpxhrU0MIRVm7QhlUeZ
T/rrl/wZJ783Mu5igK0YlQJAUSha0ZqnYLpslINEsL3Yi5uf2wUMHtb1ZJeTRv8f
D3jq1epMuqqNJ0RoW2STtwj1cQs8gcl9t4KLcJLvRyoQTQ==
-----END RSA PRIVATE KEY-----
```

上述參數可歸納如下：(請參考第四章說明)

- 模數 (modulus)：即是 RSA 演算法的參數 n。
- 公開指數值 (publicExponent)：公開鑰匙的指數 e，內定值為 65537。
- 私有指數值 (privateExponent)：私有鑰匙的指數 d，經由 RSA 演算法產生。
- 質數 1 (prime1)：即是參數 p。
- 質數 2 (prime2)：即是參數 q。

- 指數 1（exponent1）：即是 ψ(p)。
- 指數 2（exponent1）：即是 ψ(q)。
- 共通有效值（coefficient）：即是 ψ(n)。

也可以由 rsaprivate.pem 檔案內取出公開鑰匙，並儲存於 rsapublic.pem 檔案，假設不直接輸入通行碼，執行後螢幕會出現要求輸入通行碼，操作如下：(可利用前範例方法，觀察公開鑰匙檔案的內容如何。)

```
H:\SecureLab\study>openssl rsa -in rsaprivate.pem -pubout -out rsapublic.pem
Enter pass phrase for rsaprivate.pem:#####          【通行碼輸入 12345】
writing RSA key

H:\SecureLab\study>dir/b rsapublic.pem
rsapublic.pem
```

5-5-4 RSA 操作命令 – rsautl

產生了 RSA 鑰匙配對之後，吾人就可以利用 rsautl 來針對文件加密、簽署或驗證的工作。基本上，rsautl 並沒有提供雜湊演算法功能，對於大量資料需要其他演算法配合才行，否則僅能處理資料長度低於鑰匙長度的文件。命令格式如下：

```
openssl rsautl [-in file] [-out file] [-inkey file] [-pubin] [-certin] [-sign] [-verify]
                [-encrypt] [-decrypt] [-pkcs] [-ssl] [-raw] [-hex-dump] [-asn1parse]
```

- -inkey file：指定取出鑰匙的鑰匙檔案。
- -pubin：與 inkey 配合，取出公開鑰匙。
- -certin：與 inkey 配合，指定鑰匙檔案是數位憑証格式。
- -keyform：指定鑰匙格式，內定值是 PEM。
- -sign、-verfy、-encrypt、-decrypt：利用鑰匙（公開鑰匙或私有鑰匙）對輸入檔案，執行簽章、驗證、加密或解密處理。
- -pkcs、-ssl、-raw：指定輸入資料補齊方式。一般要求輸入資料的長度與加密鑰匙相同，加密後密文輸出也與鑰匙相同長度。但許多情況不會剛好如此，輸入資料太短需補齊與鑰匙相同長度，太長需分割成若干區段，最後那個區段也需補齊。但補齊方式也

有： –pkcs 與 –ssl 表示輸入資料少於鑰匙長度 11 字元，-raw 為不執行補齊操作，但三者密文輸出都與鑰匙相同長度。

- -hexdump、-ans1parse：將輸出訊息以 16 進位碼表示或 ASN.1 (DER 編碼) 解析格式顯示。

5-5-5 RSA 加密與簽章範例

假設利用之前範例產生了 rasprivate.pem 與 rsapublic.pem 兩支 RSA 配對鑰匙，接著吾人利用公開鑰匙向明文 data.txt 加密後，得到 cipher.txt。再利用私有鑰匙解密得到 plain.txt，接著比較 data.txt 與 plain.txt 兩檔案內容是否相同，如此可驗證 RSA 演算法的操作是否正確。

- 步驟 1：顯示原明文檔案內容。

```
H:\SecureLab>type data.txt
01234567890123456789012345678 9
```

- 步驟 2：利用公開鑰匙加密，得到密文 cipher.txt。

```
H:\SecureLab>openssl rsautl -in data.txt -out cipher.txt -inkey rsapublic.pem -pubin -encrypt
Loading 'screen' into random state - done
H:\SecureLab>dir/b cipher.txt
cipher.txt
```

- 步驟 3：利用私有鑰匙解密，得到明文 plaint.txt。

```
H:\SecureLab>openssl rsautl -in cipher.txt -out plain.txt -inkey rsaprivate.pem -decrypt
Loading 'screen' into random state - done
Enter pass phrase for rsaprivate.pem:####          【輸入私鑰的通行碼】
```

- 步驟 4：觀察 plaint.txt 與 data.txt，兩檔案內容是否相同。

```
H:\SecureLab>type plain.txt
01234567890123456789012345678 9
```

吾人再利用 Linux 系統，執行一個 RSA 簽署資料範例。首先 rsa 命令利用私有鑰匙（rsaprivate.pem）簽署明文檔案 data.txt，OpenSSL 會將簽署碼與原明文輸出到指定檔案上（sign.txt）。假設 sign.txt 傳遞給對方之後，對方欲證明該檔案並非他人偽造，則利用發送端的公開鑰匙(rsapublic.pem)，驗證簽署檔案，如果成功的話，則將原明文內容寫入 plain.txt 檔案內。操作如下：

```
[tsnien@csu_linux study]$ openssl rsautl -sign -inkey rsaprivate.pem -in data.txt -out
sign.txt [tsnien@csu_linux study]$ ls -l sign.txt
Enter pass phrase for rsaprivate.pem:                    【輸入通行碼】

-rw-rw-r-- 1 tsnien tsnien 64    8 月    5 14:06 sign.txt
[tsnien@csu_linux study]$ openssl rsautl -verify -pubin -inkey rsapublic.pem -in sign.txt
-out plain.txt
[tsnien@csu_linux study]$ cat plain.txt
01234567890123456789012345678 9
```

5-6 DH 公鑰演算法操作

DH（Diffie-Hellman）演算法提供一套交換鑰匙材料的公開鑰匙系統，許多應用系統利用它產生僅使用一次的會議鑰匙。DH 演算法僅產生通訊用的會議鑰匙，並不進行加密或解密的動作(相關原理請參考第四章)。OpenSSL 提供有 gendh、dhparam 與 dh 等三只命令，以下分別介紹之。

5-6-1 產生鑰匙材料 - gendh

DH 演算法參數有原數 g 與模數 n（即是 $g^{xy} \bmod n$ 中的 n 與 g），利用 gendh 產生的命令格式如下：

```
openssl gendh [-out filename] [-outform DER | PEM] [-2 | -5] [-rand file(s)]
                [-engine id] [numbits]
```

- -out filename：指定參數輸出檔案的名稱。
- -outform DER |PEM：指定輸出檔案的編碼格式，內定值為 PEM。
- -2 | -5：DH 最主要是產生共用模數 n，至於原數 g 可以是 2 或 5，一般也會採用 3，

但 OpenSSL 只使用 2 和 5。內定值是 2。

- -rand file：指定亂數產生的質子（seed）。
- -engin id：產生其他指定系統的格式。
- numbits：產生鑰匙參數（n）的長度，一般都是 512 或 1024 位元。

操作範例。吾人產生一個 512 位元長度的鑰匙參數，g 指定為 5，輸出儲存於 dh512.pem 檔案內，操作如下：

```
H:\SecureLab\study>openssl gendh -out dh512.pem -5 512
Loading 'screen' into random state - done
Generating DH parameters, 512 bit long safe prime, generator 5
....
```

5-6-2 管理 DH 命令 – dh

基本上，所產生的 DH 參數檔案並不需要經過加密處理，因為參數內容都是公開的，但有時候也需要一些適當處理，命令格式如下：

```
openssl dh [-inform DER | PEM] [-outform DER | PEM] [-in filename]
        [-out filename] [-noout] [-text] [-C] [-engine id] [-check]
```

- -inform、-outform：指定輸入與輸出檔案格式，內定值為 PEM。
- -in、-out：指定輸入與輸出檔案名稱。
- -noout：不輸出到輸出檔案。
- -text：將 DH 參數檔案以明文解析。
- -C：將 DH 參數檔案轉換成 C 語言格式。

操作範例：吾人將之前產生的 dh512.pem 參數檔案以明文格式，顯示於螢幕上。操作如下：(內容請參考第四章說明)

```
H:\SecureLab\study>openssl dh -in dh512.pem -text
Diffie-Hellman-Parameters: (512 bit)
    prime:
        00:c5:8c:8c:0a:94:a5:be:7d:2d:84:b0:28:08:59:
        0c:a4:d8:07:62:f4:74:56:86:65:4e:94:db:97:7f:
```

```
            0f:04:44:07:bb:bc:36:d8:6c:fe:01:eb:7f:02:05:
            80:1a:59:df:c1:55:64:76:57:c6:44:ac:82:14:29:
            09:7e:21:8b:e7
    generator: 5 (0x5)
-----BEGIN DH PARAMETERS-----
MEYCQQDFjIwKlKW+fS2EsCgIWQyk2Adi9HRWhmVOlNuXfw8ERAe7vDbYbP4B6
38C
BYAaWd/BVWR2V8ZErIIUKQl+IYvnAgEF
-----END DH PARAMETERS-----
```

- 質數（prime）：即是 DH 參數的模數 n。
- 產生數（generator）：即是 g 共用參數。

5-6-3 管理 DH 參數 - dhparam

OpenSSL 較新版本已將 dhgen 與 dh 兩命令功能整合於 dhparam 內，吾人僅介紹他所增加的命令。命令格式如下：

```
openssl dhparam [-inform DER | PEM] [-outform DER | PEM] [-in filename]
        [-out filename] [-dsaparam] [-noout] [-text] [-C] [-2] [-5] [-rand
        file(s)] [-engine id] [numbits]
```

- -dsaparam：產生 DSA 樣式的 DH 參數。

操作範例：吾人欲產生一個 DSA 樣式的 DH 參數，並將 PEM 格式轉換成文字格式顯示於螢幕上，操作如下：

```
H:\SecureLab\study>openssl dhparam -dsaparam -out dh_dsa512.pem 512
Loading 'screen' into random state - done
Generating DSA parameters, 512 bit long prime
.....+....+.+.............+.........+....+.....................................
.+........+......++++++++++++++++++++++++++++++++++++++++++++++++++++++++++*
.+.+.....................+....+...+...+......+....+.....+.+....+.............
..+.+......+....+.............+.......+..........+....++++++++++++++++++++++++++
++++++++++++++++++++++++*
H:\SecureLab\study>openssl dhparam -in dh_dsa512.pem -text
```

```
Diffie-Hellman-Parameters: (512 bit)
    prime:
        00:81:4e:c9:58:bc:33:92:8f:64:3a:d4:d9:db:d8:
        9d:75:e2:70:5a:1d:e6:bd:d3:0b:72:bb:6a:f4:87:
        37:7f:74:79:b4:de:94:b9:58:3d:a4:1d:84:82:a0:
        01:88:2a:aa:c4:f8:da:01:0f:7f:06:1d:91:a4:de:
        22:e0:f4:dd:79
    generator:
        7d:72:91:16:db:05:d1:a3:05:c6:54:71:6b:ae:95:
        1d:af:d0:72:f2:9e:19:6a:ae:cd:4e:40:14:21:e0:
        cf:22:c4:95:33:d9:01:f5:a0:6c:27:93:a2:d0:9d:
        95:a3:96:94:19:b9:4c:6f:8e:b9:0d:33:8c:20:ed:
        50:24:ea:4a
    recommended-private-length: 160 bits
-----BEGIN DH PARAMETERS-----
MIGJAkEAgU7JWLwzko9kOtTZ29iddeJwWh3mvdMLcrtq9Ic3f3R5tN6UuVg9pB2E
gqABiCqqxPjaAQ9/Bh2RpN4i4PTdeQJAfXKRFtsF0aMFxlRxa66VHa/QcvKeGWqu
zU5AFCHgzyLElTPZAfWgbCeTotCdlaOWlBm5TG+OuQ0zjCDtUCTqSgICAKA=
-----END DH PARAMETERS-----
```

5-7 DSA 公鑰演算法操作

DSA（Digital Signature Standard）是美國國家標準的數位簽章演算法，請參考本書第七章介紹。DSA 系統除了具有產生公開鑰匙配對外，也提供簽署與驗證功能。OpenSSL 也相對提出了 gendsa、dsa 與 dsaparam 三只命令，以下分別說明之。

5-7-1 產生 DSA 鑰匙參數 – dsaparam

DSA 鑰匙參數包含有 p、q 與 g（請參考第七章介紹），無論產生鑰匙配對、簽署文件或驗證都需使用到這 3 個參數，而且它必須公開的，又稱為公共鑰匙。許多應用系統，將某些固定區域之間通訊都採用相同的公共鑰匙，來產生不同的鑰匙配對，如此也不會影響系統的安全性。產生 DSA 鑰匙參數後，可利用 gendsa 命令產生鑰匙配對，或再利用 dsaparam 命令產生亦可。命令 dsaparam 格式如下：

```
openssl dsaparam [-inform DER | PEM] [-outform DER | PEM] [-in filename]
        [-out filename] [-noout] [-text] [-C] [-rand file(s)] [-genkey]
        [-engine id] [numbits]
```

- -inform、-outform：指定輸入/輸出檔案的編碼格式。
- -in、-out：指定輸入/輸出檔案名稱。
- -noout、-text、-C：轉換輸入檔案成另一種資料型態。
- -rand：指定亂數產生因子。
- -genkey：產生公開鑰匙配對（取代 gendsa 功能）。
- numbits：鑰匙位元數，一般 512 或 1024。

操作範例:吾人欲產生一套 512 位元的 DSA 鑰匙參數,並儲存於 dsa512.pem 檔案內。接著再觀察這些參數以 C 語言的宣告語法（其它鑰匙檔案也可利用此方法觀察其資料結構）。命令輸入如下：

```
H:\SecureLab\study>openssl dsaparam -out dsa512.pem 512
H:\SecureLab\study>openssl dsaparam -in dsa512.pem -C -noout
static unsigned char dsa512_p[]={
        0xBE,0xFC,0x35,0xAB,0x5B,0xEC,0x03,0x7C,0x17,0xD6,0xE9,0xEF,
        0x7C,0xAC,0x63,0xA3,0xCF,0xE8,0xEE,0x20,0xDF,0xDB,0xE9,0x45,
        0xE5,0x3D,0x30,0x61,0x5B,0xF5,0xD0,0x33,0x98,0x56,0xE4,0xC1,
        0x7D,0x17,0xEF,0xA6,0xFE,0x42,0x18,0x6B,0xC0,0xAC,0x86,0x31,
        0x57,0xBB,0x8D,0xD1,0x31,0xC6,0x4C,0x8B,0xFC,0x7A,0xBD,0x96,
        0x8C,0x2A,0xCF,0x01,
        };
static unsigned char dsa512_q[]={
        0xED,0xA9,0x17,0x06,0xF3,0x6A,0x06,0x83,0xB2,0x07,0x8E,0xAA,
        0x4E,0x5F,0x98,0x5E,0x1E,0x9C,0xDA,0x83,
        };
static unsigned char dsa512_g[]={
        0x5D,0xCE,0x41,0x7A,0x75,0x03,0x06,0xD9,0x9F,0x2D,0x0F,0x99,
        0xB5,0x09,0xE6,0xDD,0x91,0xE8,0xB5,0x9C,0x13,0x26,0x78,0x47,
        0xAD,0x32,0x25,0xC1,0x71,0x27,0x33,0xE5,0x86,0x68,0xCD,0x36,
        0x7A,0x3B,0x15,0xFF,0x7F,0xB1,0x1A,0x61,0x58,0xF7,0x48,0xA1,
```

```
        0xB3,0xBF,0x08,0xE7,0x8F,0x33,0xD5,0xF9,0x77,0x71,0x6E,0x5E,
        0x66,0x9B,0xA9,0x9E,
        };
DSA *get_dsa512()
        {
        DSA *dsa;

        if ((dsa=DSA_new()) == NULL) return(NULL);
        dsa->p=BN_bin2bn(dsa512_p,sizeof(dsa512_p),NULL);
        dsa->q=BN_bin2bn(dsa512_q,sizeof(dsa512_q),NULL);
        dsa->g=BN_bin2bn(dsa512_g,sizeof(dsa512_g),NULL);
        if ((dsa->p == NULL) || (dsa->q == NULL) || (dsa->g == NULL))
                { DSA_free(dsa); return(NULL); }
        return(dsa);
        }
```

產生上述 C 語言的資料結構後，即可連結到應用系統程式上，OpenSSL 即是利用此方法，將產生的鑰匙參數與應用結合。同樣的，吾人也可觀察所參數的共用參數如何，操作如下：

```
H:\SecureLab\study> openssl dsaparam -in dsa512.pem -text
DSA-Parameters: (512 bit)
    p:
        00:8f:e3:07:dc:9c:2f:39:bc:5d:1e:a4:e2:f4:60:
        82:1b:b4:e9:2c:4e:75:f3:11:62:19:40:8c:d4:67:
        ec:06:65:da:72:05:fc:cd:70:5d:03:b1:55:1a:ae:
        bb:31:90:53:0b:f0:cb:d5:2a:9e:0f:d0:89:f7:38:
        0e:f2:3b:e1:99
    q:
        00:a1:97:5f:02:14:a2:f5:c4:fd:bc:24:ff:ed:c8:
        9d:9b:2b:f9:05:89
    g:
        53:ab:86:95:27:0f:84:f4:ea:4a:8c:bf:23:8b:16:
        34:10:1f:6d:71:58:e9:04:53:dc:3f:c3:4b:58:2d:
```

```
            6d:61:8d:fe:2e:8e:44:73:d4:00:4d:94:3f:62:6e:
            60:f2:8c:c9:1a:e9:71:41:75:b1:e5:f4:89:fd:af:
            1f:44:39:af
-----BEGIN DSA PARAMETERS-----
....
-----END DSA PARAMETERS-----
```

由上述檔案內容可以看出，利用 dsaparam 命令所產生的共用參數 p、q 與 g。一般組織單位大多僅採用一套共用參數，組織內所使用的鑰匙配對，都是利用同一共用參數（或稱共用鑰匙）所產生。

5-7-2 產生 DSA 鑰匙配對 – gendsa

得到 DSA 參數之後，則可利用 gendsa 來產生 DSA 公開鑰匙與私有鑰匙，其命令格式如下：

```
openssl gendsa [-out filename] [-des] [-des3] [-idea] [-rand file(s)]
        [-engine id] [paramfile]
```

- -out filename：輸出儲存公開鑰匙與私有鑰匙的檔案名稱。
- -des、-des3、-idea：向私有鑰匙加密的演算法。
- paramfile：輸入 DSA 參數檔案之名稱，事先利用 dsaparam 命令產生。

操作範例：吾人利用 dsaparam 產生鑰匙參數檔案後（dsa512.pem），再利用它產生 DSA 鑰匙配對，並儲存於 dsa512.key 檔案內，其中私有鑰匙經過 AES-256 演算法加密，加密鑰匙是利用通行碼 12345678，操作如下：

```
H:\SecureLab>openssl gendsa -out dsa512.key -aes256 -passout pass:12345678 dsa512.pem
```

吾人可觀察 dsa512.key 檔案內容，如下：

```
H:\SecureLab\study>type dsa512.key
-----BEGIN DSA PRIVATE KEY-----
Proc-Type: 4,ENCRYPTED
DEK-Info: AES-256-CBC,9A5FE4253DFA98444F115F71B29029BA
```

```
ORQkpmSxqRm3NRpGUZ7raK7O5E3BsLgFOCKwnO+0fs7qF2aAayo3bVG9NsZg5okl
Lj2d+E6cfkH6r2ml2FP4pTZZt/A+lltpbj4+CIaS0Hxhg7KzkX79ql6gMwEedo3Z
hFkfwAHKBF8E26dWFaGmzjdoGGLZk/oVEk5u18274evaWkSoq+qyKSOCjTUR9ZwL
B/OaDXuR6FwyLwk2alBlNmaOVqiJvYNUmOA941+9Y44SJfJtxNRkx5KUMl20ODQH
oTAGjKoxKUJkURTxwsGLwaNEXzznYtSwq/RRA5+PILivS+r9TpyWH3AeGBYA+IyQ
GjbPVdi788J0LxSnuxkUiw==
-----END DSA PRIVATE KEY-----
```

5-7-3 管理 DSA 鑰匙 - dsa

OpenSSL 製作了 dsa 命令用於管理 DSA 簽署系統，譬如變更私有鑰匙加密演算法或通行碼、改變鑰匙檔案格式、等等相關操作。命令格式如下：

```
openssl dsa [-inform PEM | DER] [-outform PEM | DER] [-in filename]
        [-passin arg] [-out filename] [-passout arg] [-des] [-des3] [-idea]
        [-text] [-noout] [-modulus] [-pubin] [-pubout] [-engine id]
```

- -inform、-outform：指定輸入/輸出檔案格式。
- -in、-out：指定輸入/輸出檔案名稱。
- -passin、-passout：前者輸入通行碼以打開私有鑰匙；後者設定私有鑰匙的加密通行碼。
- -pubin、-pubout：輸入或輸出公開鑰匙。
- -text、-noout、-modulus：解析訊息輸出選項。Text 轉換文字格式；noout 不輸出編碼的鑰匙資料；modulus 輸出 DSA 公開參數。

操作範例：在前面範例中，已經產生了 dsa512.key 私有鑰匙，吾人在利用它產生相對應的公開鑰匙，操作如下。

```
H:\SecureLab>openssl dsa -in dsa512.key -pubout -out dsapub.key -passin pass:12345678
```

觀察公開鑰匙（dsapub.key）之內容如下：

```
H:\SecureLab\study>type dsapub.key
-----BEGIN PUBLIC KEY-----
MIHxMIGoBgcqhkjOOAQBMIGcAkEAvvw1q1vsA3wX1unvfKxjo8/o7iDf2+lF5T0w
```

```
YVv10DOYVuTBfRfvpv5CGGvArIYxV7uN0THGTIv8er2WjCrPAQIVAO2pFwbzagaD
sgeOqk5fmF4enNqDAkBdzkF6dQMG2Z8tD5m1Cebdkei1nBMmeEetMiXBcScz5YZo
zTZ6OxX/f7EaYVj3SKGzvwjnjzPV+Xdxbl5mm6meA0QAAkEAkeuPWqH2YCF2RpHr
Q0Wb1gdnALnqA8dOuO47c+fYyOZe59CBfBAmTczuUaMvjDiSfnKS3AgEUpZYvLDY
D5vJdg==
-----END PUBLIC KEY-----
```

操作範例：吾人想修改 dsa512.key 的加密通行碼（原來是 12345678），重新設定為 2468123，操作如下：

```
H:\SecureLab\study>openssl dsa -in dsa512.key -passin pass:12345678 -out newdsa.key -passout
pass:2468123
```

5-8 訊息摘要操作

訊息摘要演算法（或稱雜湊演算法）是將輸入訊息轉換成一份不可逆的訊息摘要（或稱雜湊值）。訊息摘要與原來訊息之間的相依性非常的高，原訊息只要稍微修改一點點，所產生的訊息摘要會變化很大。另一方面，很難找出兩份不同的訊息，可以計算出相同的訊息摘要，因此有稱為『數位指紋』。原訊息的長度可以任意大小，但產生訊息摘要的長度是固定的。訊息摘要演算法再配合公開鑰匙系統，則可作為訊息的數位簽章與驗證。

首先觀察 OpenSSL 提供有哪些訊息摘要演算法，如下：

```
H:\SecureLab\study>openssl/?
….
Message Digest commands (see the `dgst' command for more details)
md2               md4               md5               rmd160            sha
sha1
……
```

上述命令不僅包含訊息摘要功能，也具有公開鑰匙系統的數位簽章與驗證功能。

5-8-1 命令格式 - dgst

執行訊息摘要命令有兩種格式，一者配合 dgst 命令（即是 dgst md2 …）；另一者僅輸

入演算法命令（即是 md2 …)，兩種方法皆可。吾人用 hash 表示各種演算法名稱，則命令格式如下：

```
openssl dgst [-hash ] [-c] [-d] [-hex] [-binary] [-out filename] [-sign filename]
        [-passin arg] [-verify filename] [-prverify filename] [-signature filename] [file…]
```

- -hash：演算法名稱，可以是 –md2、-md4、-md5、-rmd160、-sha 或 –sha1。
- -c、-d、-hex、-binary：訊息摘要輸出格式，可以是字元（c)、BIO 訊息（d)、十六進位碼（hex）或二進位碼（binary）等格式。
- -sign filename : 該檔案(filename)儲存預定簽署文件的私有鑰匙。該鑰匙可能是 RSA 或 DSA 私有鑰匙，則相對應採用鑰匙的演算法處理。
- -verify filename：該檔案（filename）存放預定驗證已簽署文件的公開鑰匙。驗證結果會顯示："Verification OK" 或 "Verification Failure"。
- -prverify filename：該檔案（filename）存放預定驗證已簽署文件的『私有鑰匙』。
- -signature filename：該檔案儲存簽署文件後的簽署碼，提供可驗證功能。
- file …：預定進行雜湊演算處理的訊息檔案，可提供一個檔案以上連結計算，但僅產生一個訊息摘要。多個檔案之間以『，』(Linux ）或『；』(Windows ）分隔。

5-8-2 操作範例

操作範例 : 吾人利用 md2 將 data.txt 檔案內資料編碼成訊息摘要，並由螢幕顯示出來，操作如下：

```
H:\SecureLab\study>type data.txt
01234567890123456789012345678 9

H:\SecureLab\study>openssl dgst -md2 data.txt
MD2(data.txt)= 358f9627407024c72c0512da058e4472
```

操作範例：吾人利用 sha1 將 data.txt 檔案內資料編碼成訊息摘要，並儲存於 data_md.txt 檔案，操作如下：

```
H:\SecureLab\study>openssl sha1 -out data_md.txt data.txt

H:\SecureLab\study>type data_md.txt
SHA1(data.txt)= 3b67544bfd2d6732fad2aabac32e692b1ae9de13
```

5-9 數位簽章與驗證操作

訊息摘要配合公開鑰匙系統操作，則可達成數位簽章與驗證功能。欲完成數位簽章須具備下列程序：

- 產生一對 RSA 或 DSA 鑰匙配對，並將公開鑰匙廣播給參與通訊者。
- 傳送端將傳輸訊息經由雜湊演算法編碼產生一個訊息摘要。
- 利用 RSA 或 DSA 私有鑰匙對訊息摘要加密（或簽署），產生簽署碼。簽署碼與原訊息一併傳送給接收端。
- 接收端收到訊息後，利用傳送者的公開鑰匙向簽章碼解密，也將所收到的訊息經過相同的湊演算法編碼得到另一個訊息摘要，如果兩訊息摘要相同的話，則表示驗證正確。

5-9-1 RSA 簽署與驗證文件

吾人利用一個簡單範例來說明 RSA 簽章文件與驗證的操作程序，如下步驟：

- 步驟 1：產生 RSA 鑰匙配對檔案（rsapriv.pem），其中包含了私有鑰匙，操作如下：

```
H:\SecureLab\sign>openssl genrsa -out rsapriv.pem -passout pass:12345 -des3 1024
```

- 步驟 2：利用 rsapriv.pem 產生公開鑰匙（rsapub.pem），操作如下：（產生 rsapriv.pem 與 rsapub.pem 鑰匙配對）

```
H:\SecureLab\sign>openssl rsa -in rsapriv.pem -passin pass:12345 -out rsapub.pem -pubout
```

- 步驟 3：利用某一種雜湊演算法（如 sha1，雙方已協議）向訊息（data.doc）計算得到訊摘要，接著利用私有鑰匙對該訊息摘要加密，得到一個簽章碼，並將儲存於 sign.txt 檔案內，操作如下：（完成後，將 data.doc 與 sign.txt 一併傳送給接收端）

```
H:\SecureLab\sign>openssl sha1 -sign rsapriv.pem -out sign.txt data.doc
Enter pass phrase for rsapriv.pem:####                    【輸入通行碼 12345】

H:\SecureLab\sign>dir/b sign.txt
sign.txt
```

➢ 步驟 4：接收端收到訊息後（data.doc），即可利用對方的公開鑰匙（rsapub.pem）驗證簽章碼（sign.txt），如果正確的話（Verified OK），則可確認該訊息是傳送端發送的，並非被偽造或遭受竄改。操作如下：

```
H:\SecureLab\sign>openssl sha1 -verify rsapub.pem -signature sign.txt data.doc
Verified OK
```

5-9-2 DSA 簽署與驗證文件

吾人還是利用一個簡單範例，說明 DSA 系統簽署與認證文件的步驟。

➢ 步驟 1：產生一組 DSA 鑰匙參數（dsaparam.pem），它可以產生多個鑰匙配對。操作如下：

```
H:\SecureLab\dsa>openssl dsaparam -out dsaparam.pem 1024
```

➢ 步驟 2：再利用 gendsa 命令由已產生的鑰匙參數，製造出一個 DSA 私有鑰匙，該鑰匙利用 des3 演算法加密，加密鑰匙是 02468。操作如下：

```
H:\SecureLab\dsa>openssl gendsa -out dsapriv.pem -des3 -passout pass:02468 dsaparam.pem
Loading 'screen' into random state - done
Generating DSA key, 1024 bits
```

➢ 步驟 3：接著利用 dsa 命令，由私有鑰匙（dsapriv.pem）製作出相對應的公開鑰匙（dsapub.pem）。操作如下：（將公開鑰匙發布給參與通訊者）

```
H:\SecureLab\dsa>openssl dsa -in dsapriv.pem -out dsapub.pem -pubout
read DSA key
```

```
Enter PEM pass phrase: #####                    【輸入通行碼】
writing DSA key

H:\SecureLab\dsa>dir/b
dsaparam.pem
dsapriv.pem
dsapub.pem
```

- 步驟 4：傳送端將所欲傳送的訊息（data.doc）經由 DSA 簽章工具 dss1 與自己的私有鑰匙（dsapriv.pem）簽署後，得到一個簽章碼並將它儲存於 sign.txt 檔案內。操作如下：（將訊息與簽章碼一併傳送給接收端）

```
H:\SecureLab\dsa>openssl dgst -dss1 -sign dsapriv.pem -out sign.txt data.doc
Enter pass phrase for dsapriv.pem:              【輸入通行碼】

H:\SecureLab\dsa>dir/b sign.txt
sign.txt
```

- 步驟 5：接收端利用對方的公開鑰匙（dsapub.pem）驗證，操作如下：

```
H:\SecureLab\dsa>openssl dgst -dss1 -verify dsapub.pem -signature sign.txt data.doc
Verified OK
```