嵌入特定資訊於 RSA 模數

楊吳泉
義守大學 資訊工程學系
wcyang@isu.edu.tw

摘要

RSA 系統是目前使用最為廣泛之公鑰密碼系統，在憑證使用以及網路安全協定都有其不可忽略的重要性。RSA 的模數 *n* 使用上一般為隨機產生，本文探討在 RSA 金鑰產生過程中，將資訊嵌入 RSA 模數 *n* 之情形，如嵌入身分認證資訊、嵌入隱藏的資訊或是嵌入可加速運算的資訊等種種可能性，是一個有趣的研究議題，可望有許多有趣的應用。一般而言，在 RSA 模數 *n* 兩個質因數 *p* 與 *q* 長度相近時，嵌入資訊的最大長度大約是模數 *n* 長度之一半。在安全性分析方面，比較不容易，本文只作初步安全考量，希望未來能有更嚴謹的分析。

關鍵詞：公鑰密碼系統、RSA 模數、金鑰產生、蒙哥馬利乘法

一、背景

公鑰密碼系統在現今具有非常重要的應用，目前使用最廣泛的便是 RSA 公鑰密碼系統[1]。因應 RSA 系統的廣泛使用，對等的因數分解技術也快速發展[2]，所需要的 RSA 模數也隨之變大，目前一般憑證所使用的 RSA 模數已可達 2048 位元[3]。針對 RSA 系統便有許多可探討地方，其中之一便是在 RSA 模數嵌入預先決定好的內容，這個方法首先由 Lenstra 提出，可以在 RSA 模數的前面幾位數、後面幾位數，或是前後個幾位數[4]。後來陸續有學者針對其應用提出探討，例如：用來做金鑰信託(key escrow)、節省金鑰儲存空間[5]，用來內嵌密鑰(secret key)[6]或是用來加速運算[7]。

公鑰密碼系統基本上是植基在單向暗門函數(trapdoor one-way function)上，因此大多數公鑰密碼系統的金鑰產生都是先產生私鑰後，再產生公鑰。因此要將特定資訊嵌入在公鑰便有一定難度。以 RSA 系統為例，先產生兩個質數 *p* 及 *q* 再將其相乘作為公開模數 *n*。將資訊嵌入在 *n* 須有配套的技術。本文主要介紹這個嵌入特定資訊的技術，並以嵌入身分資訊及加速運算作為應用範例，文章架構如下：第貳章探討如何嵌入特定資訊於 RSA 模數；第參章介紹一些相關應用與安全考量；第肆章作個結論。

二、嵌入特定資訊於 RSA 模數

RSA 系統是大家應該都很熟悉的公鑰密碼系統，我們專注於 RSA 模數之產生，因此先將 RSA 金鑰產生過程概述於下:

1. 產生兩個隨機大質數 p 與 q。
2. 計算 $n = p \times q$ 以及 $\phi(n) = (p-1) \times (q-1)$。
3. 產生隨機亂數 e，$\gcd(e, \phi(n)) = 1$。
4. 計算 d ， $ed = 1 \bmod \phi(n)$。
5. 以(e,n)為公鑰，(d,n)或(d,p,q)為私鑰。

因為先產生 p 及 q，因此如果要嵌入資訊在 n，基本上至少必須控制一個質數的內容，使得 n 的內容可以得到控制。另一方面也得確保被控制的質數，真的是質數，RSA 始能正確運作。Lenstra 所提出來的演算法目標，便是將一個預先決定好的內容，嵌入到 RSA 模數之中，其位置可以在模數的前端、後端或是在前後端都置入資訊[4]。Lenstra 的演算法主要調整 RSA 的模數，即為調整 RSA 金鑰產生過程步驟 1 中，隨機產生 p、q 的動作，為了方便說明，我們先先對符號做扼要說明：假設要嵌入模數 n 的資訊是 s，n 的其餘部分為 r。則三種情況可以分別表成 $n = s \| r$ 、 $n = r \| s$ 及 $n = s_1 \| r \| s_0$，其中 $\|$ 表示串接的意思。此外，因為n很大，n可以下列方式表示，$|n|$表示 n 的長度，其餘整數亦相似。

$$n = \sum_{i=0}^{|n|-1} n_i d^i \tag{1}$$

其中，d 可以視為數的表示基底，其大小可視計算機實際情況來決定，例如：32-bit 系統，可以挑選$d = 2^{32}$。本文為方便了解，範例以 10 進制系統為之，亦即$d = 10$。以下針對嵌入的方式逐一介紹。

（一）將資訊嵌入在模數前端: $n = s \| r$

在一般 RSA 金鑰產生過程中，一樣先產生一個質數 p，接著便要控制 q 的內容，使得$n = p \cdot q$的前端內容為 s，詳細步驟參考圖 1。

I/P: s
O/P: $n = pq = \ s \| r$為符合 RSA 系統之模數

Step:
1. 選擇一隨機質數 p;
2. 計算 $q' = \left\lceil \frac{s \cdot d^{|r|}}{p} \right\rceil$;
3. 選擇 t，使得$q = q' + t$ 為質數 ;
4. 計算 $n = p \cdot q$;
5. 檢查 $n = \ s \| r$是否成立，若否回到步驟 1;
6. 輸出 n ;

圖 1: 將特定資訊 s 嵌入 n 之前端

舉例來說，若要產生一 8 位數的 RSA 模數，其前兩位數字為 53，可以先隨機產生

一質數 p，假設 $p = 6067$ ，接著依照圖 1 步驟可得如下結果 。

$$p = 6067 \rightarrow q' = \left\lceil \frac{53000000}{6067} \right\rceil = 8736 \rightarrow q = 8737 \rightarrow n = \mathbf{53}007379$$

（二）將資訊嵌入在模數後端: $n = r \| s$

與上述情況類似，一樣先產生一個質數 p，接著便要控制 q 的方式便要做調整，使得$n = p \cdot q$的後端內容為 s，詳細步驟參考圖 2。

I/P: s
O/P: $n = pq = \ r \| s$為符合 RSA 系統之模數

Step:
1. 選擇一隨機質數 p;
2. 計算$q' = k \cdot d^{|s|} + \left(s \cdot p^{-1} \bmod d^{|s|}\right)$，
   k 是長度為$(|q| - |s|)$之亂數;
3. 選擇 t ，使得$q = q' + t \cdot d^{|s|}$為質數;
4. 計算 $n = p \cdot q$;
5. 檢查 $n = \ r \| s$是否成立，若否回到步驟 1;
6. 輸出 n ;

圖 2: 將特定資訊 s 嵌入 n 之後端

舉例來說，若要產生一 8 位數的 RSA 模數，其後兩位數字為 53，可以先隨機產生一質數 p，假設 $p = 6067$ ，接著依照圖 2 步驟可得如下結果 ：

$$p = 6067 \rightarrow q' = 3459 \ \ (k = 34) \rightarrow q = 3559 \rightarrow n = 21592\mathbf{53}$$

（三）將在模數前後端都嵌入資訊: $n = s_1 \| r \| s_0$

將資訊同時嵌入 n 的前後，Lenstra 提出兩種控制方式，簡化來看，可以視為上述兩種方法的綜合：一樣先產生一個質數 p，控制 q 的方式視為前兩種方式之綜合，使得$n = p \cdot q$的前後端內容為分別為 s_1 及s_0，詳細步驟參考圖 3。

I/P: $s = s_1 \| s_0$
O/P: $n = pq = \ s_1 \| r \| s_0$ 為符合 RSA 系統之模數

Step:

1. 選擇一隨機質數 $p = p_1 \| p_0$;
2. 計算$q' = q_1 \| q_0$，其中
   $q_0 = k \cdot d^{|s_0|} + \left(s_0 \cdot p^{-1} \bmod d^{|s_0|}\right)$， k 是$(|q_0| - |s_0|)$位數之亂數;
   $q_1 = \left\lceil \frac{s_1 \cdot d^{|q_1|+|s_0|}}{p} \right\rceil$;
3. 選擇 t ，使得$q = q' + t \cdot d^{|s_0|}$為質數;
4. 計算 $n = p \cdot q$;
5. 檢查 $n = s_1 \| r \| s_0$是否成立，若否回到步驟 1;
6. 輸出 n ;

圖 3: 將特定資訊$s = s_1 \| s_0$同時嵌入 n 之前後端

舉例來說，若要產生一 8 位數的 RSA 模數，其前面數字為 5 後面數字為 3，可以先隨機產生一質數 p，假設 $p = 6067$ ，接著依照圖 3 步驟可得如下結果 。

$$p = 6067 \rightarrow q' = 8459 \ \rightarrow q = 9539 \rightarrow n = \mathbf{5}18061\mathbf{13}$$

$$\left(q_0 = 50 + \left(\frac{3}{67} \bmod 10\right) = 59,\ q_1 = \left\lceil \frac{5000}{60} \right\rceil = 84\right)$$

三、嵌入資訊於 RSA 模數應用及討論

在 RSA 模數中嵌入特定資訊有許多應用及值得探討的地方，此處針對嵌入身分資訊、加速運算、嵌入資訊長度限制及安全性考量等作探討。在不影響一般性下，方便起見，我們針對將資訊嵌入在前端的情況作探討與說明。

（一）嵌入身分資訊

目前公鑰使用者的身分認證普遍藉由憑證管理中心來做驗證，相關之簽章始具有法律效力。但是簽章可以有效證明使用者身分的情況下，許多場合也會考慮使用簽章方式驗證身分，例如某些無線網路身分認證的情況，在模數有嵌入特定資訊情況下便有優點。設想某一使用者的識別號是123456789，RSA的模數是1024位元，依照上一節的方式可以得到的p、q、n分別如下：

p = 7833 3964290259 7152785744 9294306466 3827885800 0695459714
6225619117 9820755753 0928848275 1073506220 0853135918
7987267394 4743174669 6631640848 7227298341 0812622033

q = 15760 3142032415 9373150350 7080997437 7173525968 5878219370
7090920182 7254762035 5649527239 7617233283 9794929158
1742708588 2231257550 5875061528 6524751864 9067530359

n = **123456789** 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000
0000000000 0000000000 0000000000 0000000000 0000000000
0000496405 2869755286 1765970144 0459049792 9347330954
8774224111 3622778728 3858971965 2481036595 3895133508
1852109607 5200355438 5785054790 1315398904 5848901426
4619799847

上例可以容易看出嵌入資訊的一個問題：如果所嵌入的資訊 s 不夠大時，中間會出現許多位數的 0 (以二進位來看亦是如此)，因此圖1步驟3中$q = q' + t$之 t 值便不適宜從1開始，可以給定一個亂數值做為起點。由於質數分配之密度可以估計為$1/\ln n$，亦即在 n 附近約每$\ln n$個整數有一個質數[8]，當然質數分布很不均勻，因此，此亂數之位元數可以預估為$\log_2 q - \log_2 \ln n - k$，其中， k 是一個預估的大小，讓質數測試時，最後出來的 n 能夠維持在我們嵌入內容。經此項調整後，我們可得到新的 p (維持不變)、q、n分別如下：

p = 7833 3964290259 7152785744 9294306466 3827885800 0695459714
6225619117 9820755753 0928848275 1073506220 0853135918
7987267394 4743174669 6631640848 7227298341 0812622033
q = 15760 3142205535 8642342963 9675374548 3161083562 2723500764
1654216326 9685533514 1610556413 7963351351 9141372680
5746840841 0291284993 3832125683 9593934330 6092792687
n = **123456789** 1356117017 3225788309 7040481795 0426281284
7511493836 3953285861 1161541551 3851524765 3725497157
2774119423 4518846155 6461494907 3220859289 4414021998
5588456731 0907271443 2271622126 0367177524 6749884442
3479107334 4851808405 5187195948 9684344928 8504319350
2830285389 7688412691 8033550944 6790865934 9020238914
9957472671

除了嵌入使用者識別外，也可嵌入認證單位的識別碼，這個識別碼甚至可以不公布(類似浮水印)，作為認證依據，也算是一個不錯的應用。

（二）加速 RSA 運算

RSA 密碼系統最主要運算為模指數運算(modular exponentiation)。提升模指數運算效能之演算法，可以概分為兩類:一是減少模乘法次數，另一為提高模乘法效能[8,9]。提昇模乘法運算(modular multiplication)之演算法中，蒙哥馬利乘法(Montgomery multiplication)是非常重要的方法[10,11,12]。扼要來說，蒙哥馬利乘法主要概念去除一般

模乘法運算中常會出現的試除法運算，而取代以容易運算的$d = \beta^l$進位，其中β可以採用我們方便運算的進位方式，例如以計算機而言，$\beta = 2^{32}$或2^{64}是個不錯的選擇，這一部分的技術目前已經非常成熟。蒙哥馬利運算的細節，可以參見參考文獻[10,11,12]，此處要點在於其與嵌入資訊之關聯。多精確度之蒙哥馬利乘法如圖 4 所示。

I/P: $\hat{a} = (a_{l-1}, \cdots, a_1, a_0)_d$, $\hat{b} = (b_{l-1}, \cdots, b_1, b_0)_d$, n
O/P: $\hat{c} = \hat{a}\hat{b}r^{-1} \mod n$

Step:

1. 設定 $\hat{c} = 0$;
2. $i = 0 \sim (l-1)$ 計算
   $$q_0 = (\hat{c}_0 + a_0 \cdot b_i)\boldsymbol{n'_0} \mod d$$
   $$\hat{c} = \frac{\hat{c} + \hat{a} \cdot b_i + q_0 n}{d}$$
3. 若 $\hat{c} > n$ 計算 $\hat{c} = \hat{c} - n$
4. 輸出 $\hat{c}$;

圖 4: 多精確度蒙哥馬利乘法運算

觀察多精確度蒙哥馬利演算法，可發現 $n'_0 = (d - n_0)^{-1} \mod d$ 。因此若是 $n_0 = d - 1$時，$n'_0 = 1$，在圖 4 的步驟 2 中，便無需計算再將$(c_0 + a_0 \cdot b_i)$與n'_0相乘，可以簡化運算，這個手法也可見於使用 Diffie-Hellman 金鑰協議之 RFC2412 [13]。初步評估這部分可以再提升蒙哥馬利運算 3%的運算[7]，。假若有一個 32 位元計算機，利用圖 2 的演算法，可以將最後 32 位元全部嵌入 1，我們也以 1024 位元 RSA 模數為例，可得以下範例 (為觀察位元，此處以 16 進為表示之)。

```
p = 95daa1fc c177c412 95fd7529 f1c6c28f 12428575 e0e4f11c
    2a5e351c 2fa1446f b865c8c6 f8f40b19 6abbc7f1 62f02bbc
    80da775e 7924d51c 2f6e430e b88dd08d
q = f8b34486 8e542afa b8b4c422 1a4ab646 5ff02ad6 54b9b7e5
    6995a94e 773f10bd d5b7f3ce 5e236b72 7e3bcd3e 58e31ffb
    185965b8 44d964fb d802b5ea 6f778dbb
n = 9194bce8 f5efc521 2b8682e3 89fc7cb9 7ec5a261 c3ba2daf
    3ef197c3 b977a1c5 661d6d63 448de954 a5fb29f0 08c06add
    7ba7ab66 c01d754d 1fea79cd 2f6f7cf2 7c054c1d 1c22d6b6
    b5aacdcd a99a6500 055e1273 b9f47ae1 3838bad0 02496a44
    94682d20 422fe1e9 057f99fe 9a5dca03 c562c578 e602684d
    7d85b0a1 ffffffff
```

（三）長度限制與安全性考量

對 RSA 模數嵌入特定資訊的長度是有所限制，以圖 1 演算法來說，$q' = \lceil s \cdot d^{|r|}/p \rceil$ 且$q = q' + t$，其中$s \cdot d^{|r|}$的長度為$|n|$，因此 $|s|$ 之最大值為$|n| - |p|$。圖 2 演算法，$q' = k \cdot d^{|s|} + \left(s \cdot p^{-1} \bmod d^{|s|}\right)$ 其中 k 是長度為 $(|q| - |s|)$ 之亂數且 $q = q' + t$，$s \cdot p^{-1} \bmod d^{|s|}$的長度需比$|q|$小，因此 $|s|$ 之最大值亦為$|n| - |p|$，與圖 1 之情況相同。圖 3 可視為圖 1 及圖 2 之組合，$|s|$ 之最大值亦為$|n| - |p|$。若p之長度為 n 之一半，則嵌入特定資訊的最大長度為 n 之一半。

嵌入特定資訊於 RSA 模數之安全性不易分析，RSA 系統安全性主要植基在因數分解上，我們就嵌入動作來考量：一般 RSA 系統的 p 與 q 是獨立產生，因此在破解上，面臨因數分解的問題；嵌入特定資訊 s 時，q 之值受到 p 的影響，不完全獨立，但在不知道 p 的情況下，似乎不影響其安全性，但須待進一步分析。保守來看，當我們嵌入的特定資訊不大時，對於 RSA 的安全性，影響更為有限。

四、結論

本文主要介紹將預先決定之資訊嵌入 RSA 模數之技術以及一些相關應用與探討，例如：嵌入身分認證資訊、嵌入加速運算資訊，這是一個有趣的研究議題，可望有許多有趣的應用。一般而言，在 RSA 模數 n 兩個質因數 p 與 q 長度相近時，嵌入的最大長度大約是模數長度之一半。在安全性分析方面，比較不容易，本文只作初步安全考量，希望未來能有更嚴謹的分析。

參考文獻

[1] K. Aoki, J. Frank, A. K. Lenstra, et.al., Factorization of a 768-bit RSA modulus (version 1.4), https://eprint.iacr.org/ 2010/006.pdf, (2010/02).

[2] D. M. Gordon, "A survey of fast exponentiation methods," *Journal of Algorithms*, vol. 27, pp. 129–146, 1998.

[3] D. J. Guan, "Montgomery Algorithm for Modular Multiplication," available in URL: http://guan.cse.nsysu.edu.tw/note/montg.pdf, Aug. 2003.

[4] M. Joye, "RSA Moduli with a Predetermined Portion: Techniques and Applications," *Advances in Information Security Practice and Experience*, LNCS. 4991, Springer Verlag, pp. 116-130, 2008.

[5] M. Kitahara, T. Nishide and K. Sakurai, "A Method for Embedding Secret Key Information in RSA Public Key and Its Application," *Advances in Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing*, pp. 665-670, 2012.

[6] D. E. Knuth, *The Art of Computer Programming*, vol 2. Seminumerical algorithms, Addison - Wesley, 1969, 2nd edition 1982, 3rd edition 1998.

[7] A. K. Lenstra, "Generating RSA Moduli with a Predetermined Portion," *Advances in Cryptology* - AsiaCrypt'98, LCNS. 1514, pp. 1-10, 1998.

[8] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computations*., vol. 44, pp. 519–521, Apr. 1985.

[9] R. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM*, vol.21, no. 2, pp.120–126, 1978.02.

[10] RFC2412, "The OAKLEY Key Determination Protocol," available in URL: http://www.rfc-editor.org/rfc/pdfrfc/rfc2412.txt.pdf, Nov. 1998.

[11] Wikipedia, *"Montgomery reduction,"* available in http://en.wikipedia.org/wiki/ Montgomery_reduction, view in Mar. 2015.

[12] 行政院研究發展考核委員會, 政府機關公開金鑰基礎建設技術規範(1.2 版)，http://grca.nat.gov.tw/download/gpki_ts_v1.2.pdf, p.70, 2005.08.

[13] 楊吳泉、黃庭軒,"以特定型態 RSA 公鑰加速蒙哥馬利乘法運算," 第 25 屆全國資訊安全會議, pp. 125-128, 2015 年 5 月。