甲、GNFS 簡介

RSA 演算法由 Rivest、Shamir、Adleman 三位於 1970 年代提出，至今屹立不搖，是最廣泛使用的公開金鑰密碼系統 (public-key cryptosystem)。以 RSA-512 為例，公鑰長度 512 位元 (bits)，亦即寫成十進位為 155 位 (decimal digits) 之大整數，且已知為兩個質數的乘積。欲破解該系統，最直接的方式為分解此數之質因數。

Carl Pomerance 於 1980 年代提出 "二次篩法" (QS – Quadratic Sieve)，於因數分解的研究上邁進一大步。1990 年代初，John Pollard 在數體 (number field) $\mathbf{Q}(\alpha) = \mathbf{Q}[x] / (x^3+2)$ 進行巧妙運算，提出新方法快速分解第七費馬數 (Fermat number) $F_7 = 2^{128} + 1$。此法推廣至分解其他 $r^e \pm s$ 形式的大數 (其中 e 和 r 是小整數)，稱為 SNFS – Special Number Field Sieve；應用於不具上述特殊形式的一般大數，則為 GNFS – General Number Field Sieve。

GNFS 包含四大步驟:

1. 多項式選取 (Polynomial Selection)
2. 篩法 (Sieve)
3. 矩陣化簡 (Matrix Reduction)
4. 開平方根 (Square Root)

前兩大步驟可以高度平行化，適合在 HP cluster 進行。後兩大步驟適合在 IBM p595 進行。SNFS 與 GNFS 僅第一步驟 "多項式選取" 不同，其後的步驟完全相同。由於 RSA 公鑰為本計劃著重之因數分解對象，其為不具上述形式之大整數；以下提及 NFS 即指 GNFS，符合相關文獻的一般用法。

完整 NFS 演算法極為複雜而龐大，且其涉及代數知識之艱深，亦遠甚於 QS。最早成功分解 RSA-512 的實作於 1999 年完成，共有 Arjen K. Lenstra 等 17 位頂尖專家參與；工程之浩大，由此可見一斑。

QS 與 NFS 的主要目標，都是尋找兩個整數 x 與 y，使得 $x^2 \equiv y^2 \pmod N$，其中 N 是欲分解的大數。在選取適當的 "因數基底" (factor base) 之後，QS 使用二次多項式並尋找夠多的多項式值，可由此基底完全分解；再藉由解線性方程組，得到適當組合成為完全平方數。

NFS 則必須分別選取整數 $\boldsymbol{Z}$ 與代數整數環 $\boldsymbol{Z}[\alpha]$的基底，並尋找大量整數對 (a, b)，使得 $a - bm$ 與 $a - b\,\alpha$ 皆為對應基底完全分解。此後再對方程組求解，獲得由其中一部份 (a, b) 所組成的集合 S，同時構成兩個不同代數結構中的完全平方數：

$$\prod_{(a,b)\in S}(a - b\,m) = y^2 \text{ for some } y \in Z \qquad \prod_{(a,b)\in S}(a - b\,\alpha) = \beta^2 \text{ for some } \beta \in Z[\alpha]$$

經由 **Z** 與 **Z**[α]的同態映射(homomorphism)，得到整數 x 與 y，進而分解 N。

NFS 為分解 $a - b\,\alpha$，並非直接分解代數整數，而是藉由分解 $a - b\,\alpha$ 生成之 prime ideal，經過適當的轉換，達到此目標。因而群論 (group theory)、環論 (ring theory)、體論 (field theory)、代數數論 (algebraic number theory) 等代數相關知識，為 NFS 必備基礎。

NFS 的四大步驟可以分開進行研究，進一步說明如後。

乙、第一步驟：Polynomial Selection

欲分解任一給定之 RSA number N，進行 NFS 第一步為尋找兩個不可約(irreducible)多項式 f_1 和 f_2，使得：

1. 在 modulo N 之下，f_1 和 f_2 有相同的根 m；

2. 可由因數基底完全分解 (smooth) 之多項式值，越多越好。

NFS 之中最耗時間與運算資源的步驟，是可以高度平行處理的第二步驟 “sieving”。不過第一步驟 “polynomial selection” 對 sieving 的效率影響極大。若多項式選得不好，將大幅增加第二步驟所需時間。

根據文獻所述，原先預估分解 RSA-155 所需 sieving 時間約為 RSA-140 七倍；但是實際上只用了四倍的時間，約為 35.7 CPU years。效率大幅增加的關鍵，就是使用較佳的演算法進行多項式選取，由此可知第一步驟 “多項式選取” 的重要性。

能夠加速 sieving 的多項式，必須兼顧 “size” 與 “root properties” 二方面特性。分解 RSA-155 的多項式選取由 Peter Montgomery 和 Brian Murphy 提出，可直接參考 Murphy 的博士論文。

已公開之 RSA-576 與 RSA-640 因數分解紀錄，皆由來自德國的 Bahr、Boehm、Franke、與 Kleinjung 完成。該團隊所使用的多項式選取技術，是 Kleinjung 的研究成果。此論文刊登於 2006 年 10 月的數學期刊，摘要如下：

Kleinjung 的方法並非全新，而是 Murphy-Montgomery 演算法之改進。NFS 所使用的多項式通常為五次，有六個係數。Murphy-Montgomery 修正次數較低的係數（二次項至常數項），使其擁有較佳 root property；Kleinjung 則是以此法為基礎，修正次數較高三個係數（五次項至三次項）。

丙、第二步驟：Sieving

篩法 (sieving) 是整個 NFS 演算法中，最耗時的步驟。它的目標是尋找夠多的數對 (a, b)，使得 $b^{\deg(f_1)} f_1(a/b)$ 和 $b^{\deg(f_2)} f_2(a/b)$ 皆可被基底完全分解 (smooth)。我們稱呼滿足上述條件的數對 (a, b) 為 “relation”。

NFS 使用的 sieving 有兩大類，稱為 Lattice Sieve 和 Line Sieve。二者可搭配使用，亦可單獨使用，以同時使用的效果較好。

(一) **Lattice Sieve**：於二維平面上之格子點 (lattice) 進行篩法。產生 relations 效率較高，但較容易漏掉可發展成 relation 之數對 (a, b)。

(二) **Line Sieve**：於直線上進行篩法。產生 relations 之效率較 lattice sieve 為低，但比較不會遺漏可發展出有效 relation 之數對 (a, b)。

“Early Abort” 為值得研究並付諸實作之技術：在篩法早期設定適當的門檻，對於未跨過門檻，進展過慢的數對 (a, b) 趁早捨棄，計算時不再觸及。

如此雖可能產生漏網之魚，太早放棄少數未來可產生有效 relation 之數對；但由於 sieving 是不斷持續進行，在新數對群進行篩法會有較高效率。比較得失後，early abort 仍然可以獲得優勢，有效節省時間與運算資源。

丁、第三步驟：Matrix Reduction

Matrix Reduction 可以再細分成 Filtering 和 Finding Dependencies 兩部份。

(一) Filtering

NFS 處理完 polynomial selection 並結束 sieving 之後，得到巨大且稀疏 (絕大部份元素是零) 的矩陣。下一步 “finding dependencies” 所需之方程組求解，其計算複雜度與矩陣的 size 和 weight 相關；filtering 即是對此巨大矩陣進行特別處理，使其 size 和 weight 較佳，以節省 finding dependencies 的時間。

(二) Finding Dependencies

在早期分解紀錄中，解方程組所使用的皆為高斯消去法 (Gaussian Elimination)。但是經由 sieving 所產生的矩陣都相當稀疏，因此可以利用疊代法 (iterative method) 解方程組，取代傳統之高斯消去法。常見的疊代法包括 Wiedemann 和 Lanczos，在較新的大數分解紀錄中，皆使用這兩種演算法與其變形。

Block Wiedemann 和 Block Lanczos 是為平行處理而產生的變形，計算複雜度與原形相近；但在同時使用多個處理器的情況下，可大幅縮減時間。有一些 NFS 專家認為：Wiedemann 演算法簡單、易於實作、且效率不比 Lanczos 差太多，因此推薦使用 Wiedemann。不過由目前已公開的分解紀錄來看，採用 Lanczos 的 NFS 實作，遠多於 Wiedemann。

戊、第四步驟：Square Root

Square Root 是 NFS 的最後步驟，計算於第三步驟所得的 $\prod_{(a,b)\in S}(a - b\alpha)$ 形式代數整數之平方根，再以同態函數對應至分解所需整數。

在 RSA 因數分解紀錄中，此步驟所佔的時間比例都相當低，但其牽涉的代數數論知識卻最多最廣，實作的程式碼也甚長。NFS 相關文獻中，多處在平方根的計算上著墨。

己、大數分解紀錄

以下是「General Purpose Factoring Records」資料的整理與分析。所謂 General Purpose，意指在此紀錄中，演算法執行時間僅與被分解數字之大小有關，而非該數之因數大小、或該數之形式。

◆ 1995 ~ 1999：RSA-130、RSA-140、RSA-155，皆採用 GNFS 演算法

	多項式選取
RSA-130	經由篩法實驗從 Scott Huddleston 提供的 14 個多項式中選出
RSA-140	Montgomery-Murphy，2000 CPU hours on 4 台 250MHz PCs
RSA-155	Montgomery-Murphy，0.40 CPU years on 250 MHz PCs

	Sieving 所需計算時間估計
RSA-130	500 mips years
RSA-140	2000 mips years
RSA-155	8000 mips years

	Lattice Sieve 因數基底個數大小（因數上限）
RSA-130	有理數端：250001（3497867）；代數端：750001（11380951）
RSA-140	有理數端：250000（3497867）；代數端：800000（12174433）
RSA-155	有理數端：16777216；代數端：16777216

	Line Sieve 因數上限
RSA-130	N/A
RSA-140	有理數端：8000000、代數端：16777215
RSA-155	有理數端：44000000、代數端：110000000

	矩陣大小 / weight
RSA-130	3504823*3516502 / 39.4 entries per column
RSA-140	4671181*4704451 / 32.36 entries per row
RSA-155	6699191*6711336 / 62.27 entries per row

	矩陣化簡步驟演算法 / 所花時間(CPU hours) / 作業環境
RSA-130	Blocked Lanczos / 67.5 / 700MB central memory on Cray-C90
RSA-140	Blocked Lanczos / 100 / 810MB central memory on Cray-C916
RSA-155	Blocked Lanczos / 224 / 2GB central memory on Cray-C916

	找平方根所花時間(CPU hours) / 作業環境 / 嘗試次數
RSA-130	49.5 per dependency / 150MHz R4400SC / 第三個成功
RSA-140	跑四組；14.2, 19, 19 / 250MHz SGI Origin 2000 / 四個中有一個失敗
RSA-155	跑四組；39.4 / 300MHz SGI Origin 2000 / 四個中有三個失敗

◆ RSA-100 ~ RSA-150 分解統計

GNFS 是目前所知最快的大數分解演算法，至今已經有相當豐富的分解紀錄，以及對其執行時間的估計。不過這些紀錄和估計是使用不同的程式、不同的電腦，因此很難直接比較，或對分解其他大數之計算量做出準確插值（interpolation）估計。以下資料節錄自日本團隊發表於 IACR ePrint Archive 網站上的報告，是在相同作業環境下，使用同一程式分解 RSA-100、RSA-110、RSA-120、RSA-130、RSA-140、和 RSA-150。

作業環境：Pentium 4 (Northwood)，2.53 GHz，FSB 533 MHz，Intel Desktop Boards D850EMV2，i850e chip set，1024RDRAM，PC800，FreeBSD 4.7-RELEASE-p13。

	Lost	**Sieving Time (sec)**	**Relations**	**Factor Base**	**Average Weight**
RSA-100	0	(1 天) 54271	190616	189614	87.425
RSA-100d4	0	55155	215271	214257	93.275
RSA-100d6	0	145247	336137	335114	100.826
RSA-110	0	(3 天) 193676	304917	303916	104.583
RSA-110d4	0	250946	384185	383169	110.156
RSA-110d6	0	461685	528397	527389	127.195
RSA-120	0	(9 天) 790138	687247	686239	147.686
RSA-120d4	0	1320933	741068	740055	139.638
RSA-120d6	0	1517888	904269	903253	154.616
RSA-130	0	(26 天) 2315347	1249886	1248880	167.301
RSA-140	0	(77 天) 6726258	1842573	1841554	189.108
RSA-150	305	(238 天) 20597260	4061097	4060083	184.604

d4 和 d6 分別表示使用次數為四和六的多項式，其餘皆為五次。分解 RSA-150 時，發生 relation 遺失；似乎在此作業環境下，150 位整數已經無法讓該程式完全順利執行。該文對實作數據所做的討論非常少，以下是我們的觀察與分析：

1. 每當 RSA number 的十進位數增加十位：

   (1) Relations 數量約增為兩倍； (2) Sieving 所需時間約增為三倍。

2. 由於 Sieving 之計算量佔 NFS 的絕大部份，一般皆以 Sieving 複雜度當作 NFS 之複雜度估計，其中 N 是欲分解的 RSA number：

$$\exp\left((1.923 + o(1))\,(\log N)^{1/3}(\log\log N)^{2/3}\right)$$

式中 log 是自然對數，以 e 為底。為簡化問題，假設 $o(1) = 0$，則以上式計算分解 RSA-100 至 RSA-155 複雜度，即為利用 $\log N = \log_2 N / \log_2 e$ 之轉換，並以 $\log_2 N$ = 330, 364, 397, 430, 463, 496, 512 代入。對應數值列表如下：

$\log_2 N$	$\log_{10} N$	複雜度	與 $\log_{10} N = 100$ 比較 (倍數)	與前數比較 (倍數)
330	100	6079855532853095	1.00	
364	110	31916231076059388	5.25	5.25
397	120	146438084943986272	24.09	4.59
430	130	624219115912155904	102.67	4.12
463	140	2493831689677497344	410.18	4.00
496	150	9405104212346417152	1546.93	3.77
512	155	17565312492499658752	2889.10	1.87

由本表可知：從 RSA-100 至 RSA-150，每增加十位，公式估計之運算量都增加為大約四至五倍，155 位約為 130 位的 28 倍。但是在上表日本團隊的實作數據之中，每多十位，運算量變成三倍左右。

此間差異，當為式中 $o(1)$ 所造成的影響。此處 $o(1)$ 與 $\log(N)$ 成反比，亦即 N 越大時 $o(1)$ 越小，故公式值應比上表的值大一些，且 N 越小則複雜度大越多。若依此調整，則彼此間的倍數關係，與上述實作結果大致相符。