多倍長整数の高速計算法

神奈川大学　後　保範
Yasunori Ushiro
Kanagawa University

1. はじめに

多倍長整数の高速計算法について、既に知られている事実の解説と今回考案した計算法について記述する。ここで、多倍長整数とは、64 ビット整数を超える整数で、10進数で20桁から数十兆桁までの値である。また、分数やπ(3.14...)などの計算も対象にする。暗号処理では256ビットから2048ビット程度の比較的短い桁数の整数を利用し、π計算の世界記録などでは数十兆桁の長い桁数の計算を利用する。最も実用性が高い多倍長整数計算は暗号処理である。特に、公開鍵暗号の RSA 暗号と楕円曲線暗号は多倍長整数の高速計算法で成り立っていると言える。多倍長整数の乗算は桁数により多種類の高速計算法が使用されており、gnu gmp[1)]は桁数に応じて、自動的に計算法を変更している。1998 年に多倍長の級数計算を考案[2)]し、2002 年に実証のために 1.2 兆桁のπ計算世界記録[3)]を達成した。今回、多倍長平方数の高速探査法(FSS)[4)]を考案しRSA暗号解読への適用方法を検討している。ここでは、既知の計算手法の解説として、多倍長整数の性質、乗算及び級数計算と公開鍵暗号方式を記述する。研究成果として多倍長平方数の高速探査法(FSS)について理論と実験結果及び、RSA 暗号解読への応用の検討状況を記述する。

2, 多倍長整数の主な性質

多倍長整数は素数か合成数かにより、性質が大きく変わる場合と変わらない場合がある。例として、$x^2=a \pmod N$の整数解 x を求めると、N が素数か合成数かにより激変する。ここで、整数 a は平方剰余で、$x^2=a \pmod N$が整数解 x を持つとする。一方、多倍長整数の乗算などは、素数か合成数かには依存せず、桁数だけに依存する。

N が 2048 ビットの整数で a が N の平方剰余となる任意の整数とする。N が素数なら 4Gh の PC で、$x^2=a \pmod N$の整数解 x を 1 秒間に数百回求めることができる。一方、N がほぼ同じビット数の合成数なら、最大級のスーパーコンピュータでも、$x^2=a \pmod N$の整数解 x を求めるのは数万年かかる。現在の RSA 暗号はこの性質を利用している。N が合成数で平方剰余となる任意の整数 a に対し、$x^2=a \pmod N$の整数解 x が求まれば、N は因数分解されてしまう。

多倍長整数は一般的に 32 ビット整数又は 64 ビット整数に m 分割して保持される。n 桁の加算、減算は計算量 O(n)の筆算の方法が使用される。乗算の計算は筆算が $O(n^2)$ で、高速剰余変換(FMT)[7]を使用すると O(n・log(n))である。FMT がいつも高速とは限らず n の大きさにより、優劣が決まるので更に多くの手法が使用されている。詳しくは 3 章で述べる。

多倍長整数の計算は計算アルゴリズムの宝庫である。計算方法により極端に演算量が違う場合が多い。定義式のままだと天文学的な計算量となるものが、計算方法を変えると PC で瞬時に計算できる例を示す。$Y=A^N \pmod N$ の計算である。N は 2048 ビット(10 進 618 桁)の整数で、A は 2 以上の任意の整数とする。定義式通りに N 回の乗算と剰余を計算すると、$O(2^{2048})$ の計算量で、どのように高速なスーパーコンピュータでも、一生の内に計算は不可能である。一方、簡単な工夫で、乗算と剰余の計算量を 4096 回以内で行うことができる。この方法を使用すると、PC でも $Y=A^N \pmod N$ を一瞬で計算できる。$A_0=A \pmod N$, $A_1=A_0^2 \pmod N$, $A_2=A_1^2 \pmod N$, . . . , $A_{2047}=A_{2046}^2 \pmod N$ と計算し、Y は N の下位ビットから順に 0 か 1 かを判定し、1 ならそれに対応する A_k を(mod N)のもとで乗算する。公開暗号方式はこの手法を利用して、暗号化、復号化を行っている。

3. 多倍長整数の乗算

多倍長整数の乗算は筆算方式から高速剰余変換(FMT)[7]まで多数の方式がある。多倍長整数の桁数を n とすると、その演算量は筆算方式が $O(n^2)$ となり、FMT では O(n・log(n))となる。中国剰余定理を使用すると、多倍長整数を分割し、分割単位で変換し、変換した後で分割したものから元の多倍長整数に戻すこともできる。ここでは、単純に多倍長整数の乗算について述べ、中国剰余定理は除く。多倍長整数の乗算を高速にするには、桁数により計算方法を変える。桁数を n とし、乗算方法を計算量のオーダーが多い順に記載する。実際の計算量はオーダーの大きさだけでは決まらない。例えば、64 ビット整数を乗算して 128 ビットの値を求めるのは、筆算方法が最も高速になる。桁数 n が大きくなるに従い、オーダーの少ないものを使用する。n 桁の乗算を O(n・log(n))で求める方法は高速フーリエ変換(FFT)が知られている。通常 FFT は共役複素数を使用するので、ここでは整数を使用して計算するものに限定する。FFT は整数で変換する FMT も含んでいる場合もある。ここでは分けて記載する。

(1) 筆算方式 : 計算量 = $O(n^2)$
(2) Karatsuba 法 : 計算量 = $O(n^{\log(3)/\log(2)}) = O(n^{1.583})$
(3) Toom 3-Way 法 : 計算量 = $O(n^{\log(5)/\log(3)}) = O(n^{1.465})$
(4) Toom 4-Way 法 : 計算量 = $O(n^{\log(7)/\log(4)}) = O(n^{1.404})$
(5) FMT(高速剰余変換): 計算量 = O(n・log(n))

筆算方式は、n 桁の多倍長整数を分割し、互いに全て乗算し、同じ桁になるもの同士を加算し、結果を桁上げ処理する。この計算量は容易に $O(n^2)$ と分かる。

多倍長整数の乗算 C=A×B を Karatuba 法で計算する方法を以下に示す。この時、A,B の桁数は同じで n 桁とする。E は A.B を n/2 桁に分割する単位とし、a_0, a_1, b_0, b_1 は n/2 桁の多倍長整数とする。

まず、A,B を半分の桁に分割する。

$$A = a_0E + a_1, \quad B = b_0E + b_1$$

次に、乗算結果 C を下記で求める。

$$C = c_0E^2 + c_1E + c_2$$

ここで、c_0, c_1, c_2 は下記で計算する。

$$c_0 = a_0 \times b_0, \quad c_2 = a_1 \times b_1$$

$$c_1 = (a_0+a_1) \times (b_0+b_1) - c_0 - c_2 \qquad \text{--- (1)}$$

c_0, c_2 は定義通りであるが、c_1 の計算に工夫がされている。c_1 は定義通りでは $c_1=a_0 \times b_1+a_1 \times b_0$ と乗算が 2 回発生する。一方、(1)式では c_1 は 1 回の乗算で計算できる。加算は増えるが、多倍長整数では乗算の計算量が $O(n^2)$ で、加算の $O(n)$ より、桁違いに多いので、この方法で演算量が削減できる。桁数 n が多い場合は加算の計算量は無視でき、$O(n^{\log(3)/\log(2)})$ の計算量となる。

Toom 3-Way と Toom 4-Way は Karatsuba 法を拡張したものである。Karatsuba 法が 2 分割で計算式を組み立てるのに対し、それぞれ 3 分割と 4 分割して乗算の計算回数を削減する方法である。Toom 3-Way と Toom 4-Way の具体的計算方法は省略する。Toom 3-Way は n 桁の多倍長整数を 3 分割して、n/3 の長さの乗算回数を 9 回から 5 回に減らす。Toom 4-Way は n 桁の多倍長整数を 4 分割して、n/4 の長さの乗算回数を 16 回から 7 回に減らす。

桁数 n が長くなると高速剰余変換(FMT)[7]が使用される。FMT による乗算の計算量は $O(n \cdot \log(n))$ である。順 FMT 変換を FMT()、逆 FMT 変換を $FMT^{-1}()$ で、FMT の各要素単位の乗算を⊗とすると、乗算 A×B は下記のようになる。

$$A \times B = FMT^{-1}(FMT(A) \otimes FMT(B))$$

FMT(高速剰余変換)では FFT(高速フーリエ変換)と同じく、直交関数よる畳み込み理論で高速化する。即ち、多倍長の乗算は全ての項のクロスタームの乗算が必要でるが、FMT(A)⊗FMT(B)では同じ位置同士の要素の乗算でよい。N 要素で直交関数にするには ω が 1 の原始 N 乗根となる ω を使用すればよい。ω が 1 の原始 N 乗根なら $\omega^{N/2}=-1$ となる。FFT は $\omega=e^{-2\pi i/N}$ とし、単位円を N 等分する複素数を使用する。一方、FMT では $\omega^{N/2}=-1 \pmod P$ となる、整数 ω と P を使用する。$P=\omega^{N/2}+1$ とすると任意の整数 P で $\omega^{N/2}=-1 \pmod P$ が成立する。通常、ω を 2 に選び、$P=2^{N/2}+1$ を利用した FMT が使用される。桁数が長くなると $P=2^{N/2}+1$ が大きくなり、(mod P)上での乗算も工夫がいる。そこで、(mod P)上での乗算も桁数

により、FMT や Karatsuba 法又はその拡張である Toom 3-Way と Toom 4-Way のどれかを選択して行う。

多倍長整数の乗算において、桁数により選択する手法の中には入らないが、FMT でω=2,P=$2^{N/2}$+1 以外で$\omega^{N/2}$=−1 (mod N)の例を参考までに表 3.1 に示しておく。このω,P を使用すると要素内は 64 ビット演算だけで計算できる。

表 3.1 P が 2^{48} の近くの原始 N 乗根となる N,ω,P の例

	N=α=2^{35}		N=β=3・2^{32}		N=γ=3^2・2^{31}	
NO	ω	P	ω	P	ω	P
1	360	8319α+1	9375	22100β+1	8426	14636γ+1
2	933	8334α+1	5549	22226β+1	226	14771γ+1
3	1714	8549α+1	9520	22254β+1	4987	14896γ+1
4	474	8756α+1	3991	22314β+1	9607	14913γ+1
5	744	8759α+1	6955	22316β+1	7806	14994γ+1

表 3.1 より同じ N で、複数の P,ωがあることが分かる。この面白いところは、複数の P で乗算を計算し、各項単位に中国剰余定理で桁数を増加させることが可能なことである。

4. 多倍長の級数計算

三角関数、指数関数、対数関数及び逆三角関数などの数学関数はテーラー展開で無限級数に展開できる。このような無限級数に展開できる関数を多倍長の精度で計算するためには、各項ごとに多倍長計算を行いそれらの和を用いる方法が知られている。この場合、n 桁の乗算の計算量を M[n]とするとき、n 桁精度の関数値計算に O(M[n]・n)の計算量が必要である。これに対し、m 桁(m≤n)の精度を持つ入力値の n 桁精度の関数計算の計算量を O(M(n)・log(n)・log(m))に削減できる方法がある。この方法は、有理数の級数に分割統治法を適用するので、分割有理数化法(Divide and Rationalize Method, DRM)[5),6)]と名づけられている。2 以上の整数 x の関数 $\tan^{-1}$(1/x)などは m が O(1)に相当し、n 桁の関数値の計算量は O(M(n)・log(n))となる。

DRM の原理は、各有理数の項数を半分に桁数を 2 倍化する通分をトーナメント式に繰り返すことである。この時、乗算の計算量は FMT を使用して M[n]=O(n・log(n))にする。以下に、DRM の原理を示す。通常は各項の分母に x の冪乗などが付くが説明を簡単にするため、省略し項数も 8 項とした。

$$f(1/x) = (A_1/B_1+A_2/B_2) + (A_3/B_3+A_4/B_4) + (A_5/B_5+A_6/B_6) + (A_7/B_7+A_8/B_8)$$
$$= (C_1/D_1+C_2/D_2) + (C_3/D_3+C_4/D_4)$$

$= (E_1/F_1+E_2/F_2) = G_1/H_1$

通分処理なので、各項は下記のようになる。

$C_i = A_{2i-1} \cdot B_{2i} + A_{2i} \cdot B_{2i-1}$,　$D_i = B_{2i-1} \cdot B_{2i}$,　$i=1,2,3,4$

$E_j = C_{2j-1} \cdot D_{2j} + C_{2j} \cdot D_{2j-1}$,　$F_j = D_{2j-1} \cdot D_{2j}$,　$j=1,2$

$G_1 = E_1 \cdot F_2 + E_2 \cdot F_1$,　$H_1 = F_1 \cdot F_2$

DRM は入力値が O(1)でなく O(m)の級数関数の値の計算も可能[8]である。一方、DRM の原理計算と同様な計算手法は、バイナリースピリット法[11]とも呼ばれる。

DRM を $\tan^{-1}(1/x)$に適用して、2002 年にπ計算の世界記録を達成[3]した。16 進数で 1 兆桁、10 進数に変換して 1.2 兆桁のπの値を求めた。それまでの記録は AGM 法(算術幾何平均法)であり、$\tan^{-1}(1/x)$による級数計算は計算量が $O(n^2)$と言われていた。DRM を使用すると計算量は $O(n \cdot \log(n)^2)$となり、AGM 法と同等になる。2002 年の記録は DRM でこれを実証した。一方 1983 年にベクトル型スーパーコンピュータ HITAC S-810/20 を用い $O(n^2)$の $\tan^{-1}(1/x)$の級数計算で、東大の金田教授と共同で 1000 万桁を達成した。これがπ計算の世界記録に計算量 $O(n^2)$の級数計算を使用した最後となった。それは、同時期に金田教授他がスーパーコンピュータより約 100 倍遅い HITAC M-280 を使用し 1600 万桁のπを AGM 法と FFT(高速フーリエ変換)を使用して達成されたためである。この時、計算方式により、計算量が大きく変わることを肌で感じた。再び、π計算で AGM 法に代わり級数計算が使用されたのは、2002 年の 1.2 兆桁である。それは DRM による計算量が $O(n \cdot \log(n)^2)$になったためである。その後、長野県の近藤茂さんと米国のアレクサンダーさんにより、5 兆桁と 10 兆桁の世界記録が達成されている。この記録では $\tan^{-1}(1/x)$より収束の速い級数公式(チュドノフスキー)に DRM が利用されている。この計算は自作 PC で最初に 16 進数の計算を行い、別公式で計算した最終の 32 桁一致することで 16 進数のπの検証を行っている。次に 16 進数から 10 進数に変換して 10 進数の記録を達成し、再度 16 進数に変換して 10 進数の検証を行っている。この進数変換にも DRM が使用されている。

π計算の最近の世界記録はスーパーコンピュータでなく PC でなされている。それには理由がある。最近のスーパーコンピュータは頭でっかちで、LINPAK での性能評価には向くが、π計算には向かない。π計算に使用する多倍長の乗算(FMT)は、LINPAK に使用される行列計算に比べ、データの再利用性が極端に低い。LINPAK のデータ再利用回数が数十万回なのに対し、π計算では数十回と万倍の差がある。そのため、多倍長整数の乗算に必要な配列が全て分散メモリに格納できても、通信性能が隘路になる。特に、最近のスーパーコンピュータは計算性能が、通信性能に比べて高過ぎ、また数万台の並列のため通信ロスも多い。更に、20 兆桁のπ計算を分散メモリ内で行う場合は、最低でも 100TB のメモリ量が必要である。数十万の超並列なら更に多いメモリが必要となる。最大級の

スーパーコンピュータでもこの量のメモリ量を確保して長時間計算は許されない状況である。このため、1回の乗算に対してもメモリ内計算を諦め、ディスクへの I/O 処理を伴う計算が必要となる。最近のスーパーコンピュータは計算速度と I/O 速度比がとてつもなく大きくなってきている。そのため、世界記録級のπ計算ではその高速計算性能はほとんど生かせない。この状況の簡単なテストとして、GPU付きの PC と、GPU無しの同一 PC で、1億桁と 100 億桁のπを計算した。その結果、メモリ内で計算できる 1 億桁は GPU 付きが無しの約 200 倍も高速に計算できた。一方メモリに収まらない 100 億桁では、GPU 付きが、GPU 無しの約 1.2 倍の高速化しか達成できなかった。ほとんどの時間が I/O 処理に費やされたためである。このため、世界記録級のπ計算は、大容量メモリに高速ディスクシステムを付けた PC がバランス的に最適と言える。

5. 現代の暗号と解読法

インターネットの発達とともに、暗号処理が重要になってくる。インターネットで情報を送受信する限り、通信データの盗聴は防げない。そのため、通信データは盗聴されても、伝えたい情報は漏れない仕組みが必要である。その役割を担うのが暗号化である。暗号化された通信データは盗聴されても、元にもどせないため、中身の情報は漏れない。現在の暗号化システムは、公開暗号鍵方式と共通鍵方式の両方を使用した、ハイブリッド暗号になっている。その理由は、共通鍵方式は暗号化、復号化が高速にできるが、肝心の鍵を交換することが不可能なためである。公開鍵方式は、安全に鍵交換ができるように数学的理論の上に作成されている。一方、公開鍵方式の処理速度は共通鍵方式ほどには高速化できない。

「http:」で始まる Web サイトは暗号化されていない。「https:」で始まる Web サイトは暗号化されている。インターネット経由でログインする Web サイトは必ず「https:」で始まることを確認する必要がある。暗号化されている Web サイトでどのような暗号方式が使用されているかは容易に確認できる。当該 web サイトを右クリックし、プロパティをクリックすると表示される。その中の「接続」に続いて表示される。現在は下記の 2 ケースが多い。

(1) TLS 1.2、AES / 256 ビット暗号 (高); RSA / 2048 ビット交換

(2) TLS 1.2、AES / 256 ビット暗号 (高); ECDH / 256 ビット交換

最初の TLS 1.2 は Transport Layer Security のバージョン 1.2 (現在最新) によるセキュリティ通信を示す。次の AES / 256 ビット暗号 (高)は、共通鍵方式に 256 ビットの AES を使用していることを示す。AES はアメリカ政府により強力な暗号方式を公募して選ばれた共通鍵暗号方式である。最後の項が公開鍵方式の暗号処理を示す。現在は 2048 ビットの RSA 暗号か 256 ビットの楕円曲線鍵交

換(ECDH)のどちらかになっている。

共通鍵方式は紀元前から使用されている。記録では、ローマ帝国皇帝シーザーが最初に使用したと言われている。この暗号は、26 文字の英字を鍵の値だけずらした簡単なものであるが、この当時は十分暗号の役割を果たした。その後種々の改良がおこなわれ、短い鍵でも頑強で暗号化、復号化共に高速にできるようになった。共通鍵方式は、暗号化と復号化に共通の鍵を使用する。共通鍵暗号方式では暗号化、復号化ともに、ビットの交換とシフトなどのビット演算の組み合わせで処理している。暗号化鍵と復号化鍵が共通なためこの方式では鍵を安全に交換、送付することは不可能である。

軍隊などでは事前に鍵を人間が持ち運び、その鍵を使用して暗号通信が可能である。一方、インターネットの発達で、だれでも容易に暗号通信する必要が発生した。そのためには、暗号鍵と復号鍵を分けた暗号方式が欠かせなかった。この要求を満たす公開鍵方式が発明されたのは、1976 年と新しい。更に具体的に実現するものとして、RSA 暗号が 1977 年にリベスト、シャミア、エーデルマンの 3 人により発明された。RSA 暗号とは 3 人の頭文字から名付けられている。RSA 暗号は多倍長整数の乗算や剰余の計算は簡単だが、合成数の因数分解が困難なことを利用している。

現在の暗号通信(暗号化、復号化)と暗号解読の計算量を表 5.1 に示す。暗号通信は高速に行え、暗号解読はどのようなスーパーコンピュータでも不可能(数千年では解けない) である。計算量は各ビット数(n)を基準にしている。ECC は楕円曲線暗号を示すが、ECDH (楕円曲線による鍵交換) 言われることも多い。

表 5.1 暗号通信(暗号化、復号化)と解読の計算量

区分	鍵の方式	暗号方式	主な計算	ビット数	計算量
暗号通信	共通鍵方式	AES	ビット演算	256	$O(n)$
	公開鍵方式	RSA 暗号	多倍長整数計算	2048	$O(n^2 \cdot \log(n))$
		ECC	多倍長整数計算	256	$O(n^2 \cdot \log(n))$
暗号解読	共通鍵方式	AES	ビット演算	256	$O(2^n)$
	公開鍵方式	RSA 暗号	リストアクセス	2048	$O(LN(a,b))$
		ECC	多倍長整数計算	256	$O(2^{n/2})$

ここで、$LN(a,b)=\exp(b \cdot (\log(n))^a \cdot (\log(\log(n)))^{1-a})$, $a=1/3, b=(64/9)^{1/3}$ である。これは、現在一番高速と言われている GNFS(一般数体ふるい法)での計算量である。n=2048 の場合、$O(LN(a,b))=O(10^{35})$ である。

RSA 暗号は下記のようにして暗号鍵、復号鍵を作成する。

(1) ランダムに素数 P,Q を作成する。P,Q は共に 1024 ビット。

(2) N=P×Q 及び f=(P-1)×(Q-1)を計算する。

(3) 整数 C をランダムに選ぶ。C は P,Q より短いビット数でよい。

(4) $d=C^{-1} \pmod f$ となる d を計算する。

このとき、C と N は暗号鍵、d と N は復号鍵となる。C,N は送信するが、d は送信しないので、盗まれる心配がない。RSA 暗号は、共通鍵の送付、交換や認証に使用される。そのため、送付される鍵長は N の長さで十分である。送付する共通鍵を R とする。送信側は R を下記で暗号化し T にする。

$$T = R^C \pmod N$$

T を受信し、もとの R に下記で復元する。

$$R = T^d \pmod N$$

$d=C^{-1} \pmod f$ で、整数 α を用いて $Cd=\alpha f+1$ と表される。また、N=P×Q で f=(P-1)×(Q-1)なので、オイラーの定理を使用すると $R^{\alpha f}=(R^f)^{\alpha}=1 \pmod N$ となる。
これより下記が成立する。

$$T^d \pmod N = (R^C)^d \pmod N = R^{\alpha f+1} \pmod N = R \pmod N = R$$

RSA 暗号の暗号化と復号化は、多倍長整数の乗算及び剰余の計算で構成される。

楕円曲線暗号(ECC)は下記の素数 p と整数 h,g で定義される曲線上を動く整数 x,y を使用する。

$$y^2 = x^3 + hx + g \pmod p \qquad \text{--- (1)}$$

(1)式の位数を r とする。r,p は強度を強くするため、共に素数に選ぶ。p が素数なので |p-r|<sqrt(p) となる。(1)式を満たす点 $S_1=(x_1,y_1)$, $S_2=(x_2,y_2)$ と $S_3=(x_3,y_3)$ に対し、二つの加法、$S_3=S_1+S_2$ と $S_3=S_1+S_1$ が定義できる。$S_3=S_1+S_2$ の計算法を下記に示す。S_2 が S_1 に一致すると別の計算法 $S_3=S_1+S_1$ となる。こちらの計算法は省略する。

$$c = (x_2-x_1)^{-1} \pmod p, \quad d = c(y_2-y_1) \pmod p$$

$$x_3 = d_2 - x_1 - x_2 \pmod p$$

$$y_3 = d(x_1-x_3) - y_1 \pmod p$$

二つの加法を組み合わせて繰り返し使用すると、整数 m の乗法 $S_3=m\times S_1$ が定義できる。このとき、$S_3=S_1+S_1$ の加法は $\log_2(m)$ 回、$S_3=S_1+S_2$ の加法は最大で $\log_2(m)$ 回使用する。ECC は共通鍵方式の鍵を交換できる。通信途中を盗聴されても、問題はない。楕円曲線上の点 S=(x,y) を A から B に送信する。この S は他人に漏れても良い。A と B がそれぞれ秘密の整数 a と b をランダムに作成する。
次に A は $S_a=a\times S$ で、B は $S_b=b\times S$ を ECC の乗法で求め、互いに相手に送信する。A は S_b を、B は S_a を受け取り、A は $a\times S_b$ を計算し、B は $b\times S_a$ を ECC の乗法で求める。ECC の乗法は交換法則が成り立ち、下記となる。

$$a\times S_b = a\times(b\times S) = b\times(a\times S) = b\times S_a$$

これで、$a\times S_b=b\times S_a$ となり、互いに共通の鍵が得られる。S_a, S_b は送信するので

漏れる可能性はあるが、a,b が秘密な限り、$a \times S_b$ や $b \times S_a$ を求めるのは計算量が多過ぎて不可能である。以下に具体的例で示す。

$y^2 = x^3 + hx + g \pmod p$ の楕円曲線とする。

ここで、h=1234567891,g=3087180044,p=4294967311 である。

S=(1357645323,727714280),a=1533275276,b=3425764634 とすると、S_a,S_b は下記のようになる。

S_a=(2233317144,3900379490)， S_b=(3288632759,711237429)

次に、$a \times S_b$ と $b \times S_b$ を計算すると共に下記となる。

$a \times S_b = b \times S_a$ = (3649365589,1798686850)

RSA 暗号の解読法として、複数多項式ふるい法(MPQS)と一般数体ふるい法(GNFS)がある。10 進 100 桁を超える因数分解は GNFS が有利で、RSA 暗号の解読には GNFS が使用されている。合成数 N を GNFS で因数分解する手順を示す。

(a) 合成数 N から多項式の係数を求める（利用関数の探査）
(b) ふるいに利用する基底（素数及び素イデアル）の選定
(c) ふるい処理で基底数以上のふるいデータの取得（ふるい処理）
(d) ふるい結果を行列にし、(mod 2)で 0-1 行列を作成
(e) 0-1 行列から従属となる行の組を求める（0-1 行列計算）
(f) 多項式を法とする代数的平方根の計算（代数平方根）
(g) $A^2-B^2=0 \pmod N$ の形に変形し、合成数 N を因数分解

現在最も長い桁数の RSA 暗号の因数分解は、2010 年 1 月に NTT 他 5 か国が共同で実行[9]した、RSA-768(10 進 232 桁)である。この因数分解は 1 次式と 6 次式の多項式からなる GNFS が使用された。表 5.2 に RSA-768 の因数分解の計算量を示す。計算量は AMD64(2.2Gh)の 1 コアで 1 年の計算を 1 台数・年で示す。

表 5.2 RS-768(10 進 232 桁)因数分解の計算量

処理項目	計算量(台数・年)	構成比率(%)
利用関数の探査	20	1
ふるい処理	1500	90
0-1 行列計算	155	9
代数平方根	1	0
その他	1	0

RSA-768 の分解に使用された、0-1 行列のサイズは、192,796,550×192,795,550 である。約 2 億次元の行列で列サイズが行サイズより 1000 だけ大きい行列である。0-1 行列で従属となる行を求めるのは、各基底のベキ数を偶数にして、平方数にするためである。表から計算量の多くをふるい処理が占めるのが分かる。

楕円曲線暗号（ECC）の解読では、現在 Pollard Rho 法（ρ 法）が最速と言われている。ECC の解読は ECDLP（楕円曲線離散化対数問題）と呼ばれて研究されている。解読は ECC の 2 つの点 S と R から R=z×S の乗法が成立する整数 z を求めることである。ECC の ρ 法による計算方法の例を示す。p は(1)式の素数 p で、r は位数であり、任意の ECC の点 S に対し、r×S は唯一の無限遠点となる。

(1) ランダムな整数 α_0, β_0 を使用し $W_0=\alpha_0\times S+\beta_0\times R$ を求める。

(2) $W_j=W_k$ まで k=1,2,3,...と(3)を繰り返し計算をする。ここで j<k である。

(3) W_k の座標を (x_k, y_k) とし、$i=x_k \pmod 3$ で 3 ケースに分けて計算する。

(a) i=0: $W_k=W_{k-1}+W_{k-1} \pmod p$, $\alpha_k=2\alpha_{k-1} \pmod r$, $\beta_k=2\beta_{k-1} \pmod r$

(b) i=1: $W_k=W_{k-1}+S \pmod p$, $\alpha_k=\alpha_{k-1}+1$

(c) i=2: $W_k=W_{k-1}+R \pmod p$, $\beta_k=\beta_{k-1}+1$

(4) $z=(\alpha_k-\alpha_j)/(\beta_j-\beta_k) \pmod r$ で整数解 z を計算する

この計算では W_k はある時点から同じ値で繰り返す。そのため、j<k となる j の W_j, α_j, β_j を全て記憶する必要はなく、一定の間隔や特徴のある場合だけ記憶すれば良い。また、上記(3)では 3 ケースではなく、8 や 16 ケース分けがよく使用される。位数が r のとき、本方式による ECC の解読の計算量は $O(2^{r/2})$ である。

6. 多倍長の平方数判定

多倍長合成数 N を P と Q に因数分解するには、$S^2=T^2 \pmod N$となる整数 S,T を求める。S 側は平方数やその積で選べるが、T 側を平方数やその積にするのが難しい。MPQS(複数多項式 2 次ふるい法)では S は平方数の積で、T は因数の各素数の積のベキが偶数になるようにする。GNFS は複雑な手順だが、最終的に $S^2=T^2 \pmod N$にする。実用性は劣るが Fermat 法では $y=(M+x)^2-N$ で平方数となる y を見つければ、N は因数分解される。ここで、M は N の平方根を整数に切り上げたもので、x は 0 から 1 ずつ増やしていく。Fermat 法では N の因数 P,Q が極端に近い場合に有効である。また N の因数 P,Q の比が整数 a,b に極端に近い場合に拡張 Fermat 法がある。拡張 Fermat 法は $y=(M+x)^2-4abN$ で、y の平方数を見つける。ここで、M は 4abN の平方根を整数に切り上げたものである。

多倍長整数の整数平方根（平方根の小数点以下を切り捨た整数）の計算は、乗算や剰余の計算に比べ遅い。一方、多倍長整数が平方数かどうかの判定は、上手く特定の剰余を使用すると、乗算や通常の剰余計算より高速になることが知られている[1)12)]。gnu gmp[1)]には多倍長整数 y が平方数かどうか判定する関数として、mpz_perfect_square_p(y)がある。これを使用すると、gmp の関数で整数平方根を計算するより、高速に y が平方数かどうか判定できる。

mpz_perfect_square_p 関数は、下記のようにして y が平方数かどうか判定している[12)]。(1)か(2)で平方数でないと分かると、判定は終了する。

(1) y の下位 1 バイト(8 ビット)を取り出し、256 ビットの平方数テーブル(平方剰余なら 1,平方非剰余なら 0)を参照して、0 なら y は平方数でないと判定する。y の下位 1 バイトは y (mod 256)に相当する。256 個の中で平方剰余は 44 個なので、83%の判定ができ(2)に進むのは 17%になる。

(2) y の mod 9,5,7,13,17 を求めてそれぞれ平方数テーブルを参照し、一つでも 0 なら y は平方数でないと判定する。この計算は $2^{24}-1=9\cdot5\cdot7\cdot13\cdot17\cdot241$ を利用して剰余(mod)の高速化を行っている。(1)(2)の判定で平方数でないと判定できない確率はわずか、0.7%である。

(3) (1),(2)で判定できなかた y に対して平方数の判定を行う。本判定では多倍長整数で y の平方根を計算し、それを 2 乗して残差がゼロなら平方数と判定する。ゼロ以外なら非平方数と判定する。

3 から 31 までの素数 P において x^2 (mod P)が取り得る値を表 6.1 に示す。これより全ての P において、取り得ない値が存在することが分かる。逆に考えると、(mod P)においてこの表にない y は $y=x^2$ (mod P)となる整数 x が存在しない。即ち、y は平方数では無いことを示している。

表 6.1 奇素数 P において x^2 (mod P)が取り得る値

P	x^2 (mod P)が取り得る値	その個数
3	0,1	2
5	0,1,4	3
7	0,1,2,4	4
11	0,1,3,4,5,9	6
13	0,1,3,4,9,10,12	7
17	0,1,2,4,8,9,13,15,16	9
19	0,1,4,5,6,7,9,11,16,17	10
23	0,1,2,3,4,6,8,9,12,13,16,18	12
29	0,1,4,5,6,7,9,13,16,20,22,23,24,25,28	15
31	0,1,2,4,5,7,8,9,10,14,16,18,19,20,25,28	16

この表から、0 を除くと x^2 (mod P)の取り得る値と、取り得ない値の数が全ての P において同じ規則に従う。取り得る値の数は(P+1)/2 となる。

また、素数 2,3,5,7 については 64 以下の冪乗した値において、x^2 (mod P)が取り得る値を表 6.2 に示す。この表から、2,3,5,7 の冪乗を P にすると、x^2 (mod P)が取り得る値の数は P の半分以下になり、素数単独より平方数でない値を取り除く効率が良いことが分かる。$P=2^6=64$ なら一回の判定で 1/5 以下に平方数候

補を絞れることが分かる。

表 6.2 素数 2,3,5,7 の冪乗において $x^2 \pmod P$ が取り得る値

P	$x^2 \pmod P$ が取り得る値	その個数
4	0,1	2
8	0,1,4	3
16	0,1,4,9	4
32	0,1,4,9,16,17,25	7
64	0,1,4,9,16,17,25,33,36,41,49,57	12
9	0,1,4,7	4
27	0,1,4,7,9,10,13,16,19,22,25	11
25	0,1,4,6,9,11,14,16,19,21,24	11
49	0,1,2,4,8,9,11,15,16,18,22,23,25,29,30,32,36,37,39,43,44,46	22

しかし、この判定方法をそのまま Fermat 法や拡張 Fermat 法に適用して、y が平方数かどうか判定して高速化するには大きな限界がある。それは、$y=(M+x)^2-N$ または $y=(M+x)^2-4abN$ で y の値を必ず計算するためである。gmp で y の計算だけと、y とその整数平方根を計算し y が平方数かどうかの判定時間を測定した。その結果、N が 1024 ビットや 2048 ビットの多倍長整数の場合、両者の速度比は約 4 倍と判明した。逆に言うと、この方法を y に直接使用すると、整数平方根を計算して平方数を判定する方法に比較して、高々4 倍しか高速化しない。そこで、y は計算しないで y の平方数候補を大幅に絞る方法を考えた。y の平方数を判定するのは、滅多に y が平方数にならず、平方数になれば問題解決する場合がほとんどである。そのため、y は $y=(M+x)^2-4abN$ のように多倍長整数 x による多項式のまま、y が平方数の判定をする方法を検討した。

その結果、y は多項式のまま、x に対応する y の値は計算しないで、y が平方数かどうか判定する方法を発見した。その方法を FSS(高速平方数探査法)と名付ける。その概要は、P に対して $x^2 \pmod P$ が取り得るテーブルの代わりに、$f(x)=(M+x)^2-N$ のような整数 x を変数とする多項式において f(x)が平方数となるテーブルを作成する。各 P に対して、x=0,1,2…として f(x)が平方数候補であるか評価する。一つの P で約半分に平方数候補は絞られるため、30 個の P を使用すると、1/十億に絞られる。2,3,5,7 は P にその冪乗を使用すれば更に絞られる。この時、$x^2 \pmod P$ のテーブルは P にだけ依存するが、多項式 f(x)のテーブルは P と f(x)に依存する。具体的な f(x)で FSS による平方数探査の計算方法を示す。$y=x^2 \pmod P$ の判定テーブルを Gp で、f(x)の平方数判定テーブルを Fp で示す。そこで使用する P と Gp テーブルを表 6.3 に示す。表 6.1、

表 6.2 と表示方法を変え、$y=x^2 \pmod N$の整数 y が 0～P−1 までのテーブルで 0 か 1 の値を持たす。整数 x で整数 y が存在すれば 1、存在しないと 0 である。

表 6.3 使用する $y=x^2 \pmod P$の判定テーブル Gp

P	Gp
8	1 1 0 0 1 0 0 0
9	1 1 0 0 1 0 0 1 0
5	1 1 0 0 1
7	1 1 1 0 1 0 0
11	1 1 0 1 1 1 0 0 0 1 0
13	1 1 0 1 1 0 0 0 0 1 1 0 1
17	1 1 1 0 1 0 0 0 1 1 0 0 0 1 0 1 1
19	1 1 0 0 1 1 1 1 0 1 0 1 0 0 0 0 1 1 0
23	1 1 1 1 1 0 1 0 1 1 0 0 1 1 0 0 1 0 1 0 0 0 0

平方数探査の例として $f(x)=(M+x)^2-N$ を使用した。N は 128 ビットの整数で、N=157222523472368979107040811720292727463, M=12538840595221273452。この場合 2 と 3 の冪乗値は 8 と 9 を使用し、P は 8,9,5,7,11,17,19,23 を使用する。P の最大値は 23 なので、f(x)の値は x が 0～22 までの 23 個を求める。次に各 P において Fp を求めるために $f(x) \pmod P$を計算する。具体的に P=8,9 で $f(x) \pmod P$を求めると下記のようになる。$f(x) \pmod P$は x が 0 から P−1 まで順に P 個求める。

P=8 : { 1, 2, 5, 2, 1, 2, 5, 2 }

P=9 : { 5, 0, 6, 5, 6, 0, 5, 3, 3 }

各 P に対して Fp のテーブルを作成するには、上記の $f(x) \pmod P$の値で、表 6.3 の Gp テーブルを使用して変換する。P=8 の場合は位置が 1,2,5 の Gp の値は 1,0,0 なので、$f(x) \pmod P$の値が 1 なら Fp の 1 で、値が 2 か 5 なら 0 に変換される。従って、P=8 の Fp テーブルは{1 0 0 0 1 0 0 0}となる。P=9 の場合は、Gp の位置 0 だけが 1 で、位置 3,5,6 はいずれも 0 なので Fp テーブルは{0 1 0 0 0 1 0 0 0}となる。P=5,7,11,13,17,19 及び 23 においても同様は方法で Fp に変換する。これらを纏めて、表 6.4 に $f(x)=(M+x)^2-N$ の Fp テーブルを示す。

表 6.4 $f(x)=(M+x)^2-N$ の Fp テーブル

P	Fp
8	1 0 0 0 1 0 0 0
9	0 1 0 0 0 1 0 0 0
5	1 1 0 0 0

7	0 0 1 1 1 1 0
11	1 1 0 1 1 0 1 0 0 1 0
13	1 0 1 1 0 0 1 1 0 1 0 1 0
17	0 1 0 0 1 1 1 1 1 1 0 0 1 0 0 0 0
19	1 1 1 0 1 0 0 1 1 0 0 0 1 1 0 0 1 0 1
23	0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 0 1 0 1 0 0 1 0 1

表 6.4 は N=157222523472368979107040811720292727463 で $f(x)=(M+x)^2-N$ の f(x)の整数平方数を探す FSS 用のテーブルである。以下具体的にその探査方法を示す。

P=8 で x が 0 から 7 までの平方数候補を Fp から選ぶと、2 件で x=0,4 が得られる。これは、8 で繰り返すので、72 未満の x では 0,4,8,12,16,20,24,28,32,36,40,44,48,52,56,60,64,68 の 18 件が得られる。この x に対し x (mod 9)は 0,4,8,3,7,2,6,1,5,0,4,8,3,7,2,6,1,5 となる。P=9 で Fp が 1 なのは 1 と 5 の位置である。これから P が 8 と 9 で平方数候補を選ぶと x=28,32,64,68 となる。同様に、P=5 の選定を追加すると 360 未満の x での、平方数候補は x=100,136,140,176,280,316,320,356 の 8 件が得られる。更に、P=7 の選定を追加すると、2520 未満の x の平方数候補は x=100,136,320,460,500,536,640, 676, 716,856,1040,1076,1180,1216,1220,1256,1360,1396,1580,1720,1760,1796,1900,1936, 1976,2116,2300,2336,2440,2476,2480,2516 の 32 件となる。

次いで、P=11,13,17,19,23 の 5 件の P を追加し、Fp テーブルを使用して 2520 未満の x で平方数候補を選ぶと、x=856,1216 の 2 件が得られる。更に次の、x が 2520 以上、5040 未満の区間で x で平方数候補を選ぶと、x=3916 が得られる。これを 14 回繰り返し 35280 までの x で平方数候補を選ぶと、x=856,1216,3916,10036,15580,27680, 29656,32000,32896 の 9 件が得られる。この 9 件にたいして $f(x)=(M+x)^2-N$ で f(x)の値を計算し、その整数平方根と残差を計算する。残差が 0 となるのは、x=32896 で f(x)が平方数になる。x=32869 とすると M+x=12538840595221306348 となり、f(x)は次のようになる。

$f(x) = (M+x)^2-N = 824958475358899372369641$

f(x)は平方数なので、その平方根を h とすると $h^2=f(x)$で、h の値は下記となる。

$h = 908272247379$

また、下記の関係式が成立する。

$h^2 = (M+x)^2 \pmod N$

従って、N=P×Q と因数分解され、P<Q とすると、P=M+x−h, Q=M+x+h となる。

これより、N は下記のように分解される。

$N = P \times Q = 12538839686949058969 \times 12538841503493553727$

この例では P と Q の値が非常に近いため、Fermat で容易に分解された。

x が 0 から順に整数平方根を求めて、f(x)が平方数か判定すると、整数平方根の計

算回数は 32870 回となる。一方、9 個の P を使用した FSS では、整数平方根の計算はわずか 9 回で済みその比は 1/3652 である。一般に FSS で m 個の素数 P を使用すると、整数平方根の計算回数が約 $1/2^m$ に減少する。今回 m=9 なので 1/512 であるが、2 と 3 は $8=2^3$,$9=3^2$ と冪乗値を使用したのでより削減効果が出ている。

整数 x を変数とする整数多項式 f(x)は、x+P=x (mod P)が成り立つので、f(x+P)=f(x) (mod P)も成り立ち、FSS が上記具体例と同様に適用できる。実際にプログラムでより高速化するには、2,3,5,7 のベキ数の選び方、繰り返し使える x の平方数候補を保存する配列の確保、そこまでに使用する P の数と順序の選定(例では 8,9,5,7 の 4 個)、使用する P の総数など種々の工夫が必要である。また、残りの P(例では 11,13,17,19,23 の 5 個)による平方数探査では、テーブルの合体や、探査順序の変更も重要である。ここでは、FSS の原理を示すため、プログラム上の詳しい工夫は省略する。7 章の実験結果はプログラム上の工夫をしたものである。

7. 平方数判定の実験結果

平方数判定の数値実験は下記の環境で行った。

PC: HP Desktop 870 (Intel Core i7 6700k), 4Gh, 8GB
システム: Windows10 Home x64
コンパイラー: Cygwin Ver. 2.7 gcc (64bit 版), 最適化:-O3
多倍長計算: gmp の mpz_t

現在の RSA 暗号と同じ 2048 ビットの N を使用し、Fermat 法と拡張 Fermat 法を対象に測定した。Fermat 法は $y=(M+x)^2-N$ である。ここで M は N の平方根を切り上げた整数である。拡張 Fermat 法は a=1017,b=1231 で $y=(M+x)^2-4abN$ である。M は 4abN の平方根を切り上げた整数である。両者共に、y が平方数になる整数 x を 0 から 1 ずつ増加して求めれば、N は因数分解される。測定は共に下記 3 方式で行った。1 番目の方式「平方根」は平方数の判定に gmp の mpz_sqrtrem 関数を使用した。mpz_sqrtrem は、整数平方根と残差を同時に求める。平方数なら残差がゼロとなる。2 番目の方式「剰余法」は gmp の mpz_perfect_square_p 関数を使用した。当該関数は、整数平方根を求める代わりに、特定の値の剰余から平方数かどうかを判定する関数である。mpz_sqrtrem 関数より平方数の判定は速い。3 番目の方式「FSS」は平方数の判定に、考案した高速平方数探査法(FSS)を使用したことを示す。この場合は y の値そのものは計算しないで、y が平方数かどうか判定可能なため、平方数の可能性がある場合だけ y の値を計算する。表中の「平方数」、「剰余法」及び「FSS」は測定した方式を示す。「FSS」は各データ共に、y が平方数になるまで計算し、N を因数分解するまでの時間を測定した。一方、「平方根」、「剰余法」の 2 方式は、y が平方数になるまでの計算は時間的に不可能なため、100 億回(x=0～100 億-1)の探査時間を測定した。表 7.1 に Fermat

法による平方数探査速度の 3 方式比較を示す。表 7.2 に拡張 Fermat 法による平方数探査速度の 3 方式比較を示す。測定データは Fermat 法、拡張 Fermat 法共に 4 件とした。Fermat 法のデータは探査回数が少ない順に A-1,A-2,A-3.A-4 とした。拡張 Fermat 法のデータも探査回数が少ない順に B-1,B-2,B-3,B-4 とした。「平方根」と「剰余法」は探査回数 100 億回で、A-1,A-2,A-3,A-4 及び B-1,B-2,B-3,B-4 において計算時間に有意差がないので、表 7.1 では A-1 を、表 7.2 では B-1 の測定値を記載した。探査数と計算時間は測定した値である。1 秒間計算数は探査数を計算時間で割り、1 秒間に何億回計算できるかを算出したものである。高速化比は、「平方根」及び「剰余法」に対する高速化比を示した。

表 7.1 Fermat 法による平方数探査速度の 3 方式比較

方式	Data	探査数（百億）	計算時間(s)	1 秒間計算数(億)	高速化比(倍)	
					対平方根	対剰余法
平方根	A-1	1	5,174	0.02	1	---
剰余法	A-1	1	1,335	0.07	4	1
FSS	A-1	900,000	135	700,000	34,000,000	9,000,000
	A-2	3,600,000	594	600,000	31,000,000	8,000,000
	A-3	14,400,000	2,499	600,000	30,000,000	8,000,000
	A-4	57,600,000	7,497	800,000	40,000,000	10,000,000

表 7.2 拡張 Fermat 法による平方数探査速度の 3 方式比較

方式	Data	探査数（百億）	計算時間(s)	1 秒間計算数(億)	高速化比(倍)	
					対平方根	対剰余法
平方根	B-1	1	5,300	0.02	1	---
剰余法	B-1	1	1,465	0.07	4	1
FSS	B-1	1,800,000	112	1,600,000	85,000,000	23,000,000
	B-2	7,200,000	561	1,300,000	68,000,000	19,000,000
	B-3	28,700,000	2,377	1,200,000	64,000,000	18,000,000
	B-4	114,700,000	5,656	2,000,000	108,000,000	30,000,000

「FSS」が超高速なのは、y の値をほとんど計算しないためである。y の値を計算するのは、平方数の可能性がある場合(1/数百億)だけである。「剰余法」の計算時間はほとんど y の計算時間である。「平方根」では y の計算時間の約 3 倍を整数平方根の計算が占め、全体で「剰余法」の約 4 倍かかる。「平方根」と「剰余法」は N のビット数が 2 倍になると、3～4 倍遅くなる。FSS は y の計算回数が極端に少ないので、計算時間は N のビット数に依存しないで、N の性質で

多少増減する。2048 ビットの N で、「FSS」は「平方根」の３千万倍～1 億倍高速になることが分かる。

測定に使用した N の値と「FSS」で N=P×Q と分解した P,Q の値を記載する。N は 2048 ビットと長いため、全ての N でなく、代表として A-3 と B-4 を示す。

(1) A-3 データの N と P,Q の値

N=25677387604979174745650113439241870319948692169987586987064217095280862955992909072300653168631621794286691440902481665333116951445888344416180968585370801121768979094149716379997854705402081724789536285696382681163337713071877854351132396919532278789858567876288171486292852151675913410606293606807062766667774100801816717926740523350168941221016784012879030675574806950125496988907696548478947055250228339395581991589705540388227820422396503216919351096257513225507347446746249892236889998843431911859366155805417959399790863966904379698649990692857919537481312400757172176581641596480966744672440739826707093583929

P=16024165377634859752393146044825983091646432053353984232099398660385403559126981008468040945868502929769482340528734701278038220200795086679953476319516238522505700232940873548567197459315687635172933081786347559217864497772533550572732717360916319335743317188215042301906000262720413551973045722380387884295 1

Q=16024165377634859752393146044825983091646432053353984232099398660385403559126981008468040945868502929769482340528734701278038220200795086679953477678729426056505456644625850175099666961389326199608523520218170085838215408395018036921856227090664904889062921478617307515748131124261700164713305389789885713087 9

(2) B-4 データの N と P,Q の値

N=25677387604979174745650113439241870319948692169987586987064217095280862955992909072300653168631621794286691440902481665333116951445888344416180966537189190379746372391927497745377928179429783605705131011667330211430085137953923275633043250509933086957796651360725933880158974798047327525684979600898227671512525576991668202280786079825714606148264571335017006371828175492589807670754064085239147879319623564388832968033123870130269924818949195272011387945323845467631856683857838803603214378972022866034185143206524925340623238795164922037435151797701445299888947195323580111950157298979476220823162503958438421236729

P=14579192959123228748927513453682387588251153484488623397204177395568532449906765935573736743521977627569135970666010533778779676888462153563971281441489633666092431002181171922134457307040675433464994978647120880037230751143990814067905111097541237847266775279225408888479337656126084371083957179091346037747 9

Q=17612351847576736594631765516666358313186624081850338961686302624087206521918772194986525938445097605041811952786907720394188206329437596699949605049290606399960703184000074992583724952788524289860578528476109231664837597119130673212083528617468191432365583648757441170066500371898858306585546957506165225075 1

「FSS」での平方数探査状況を表 7.3 に示す。「FSS」では 32 個の素数(P)を使用し、各 P での y (mod P)の組み合わせで平方数候補を選び、選ばれた y の整数平方根を求めて、平方数を判定した。「FSS」では事前準備として、P の最大値まで y の値を計算し、各 P に対し y (mod P)を求めテーブルを作成する。その後、x を増加させながら、y の平方数候補を選ぶ。表 7.3 の探査数及び候補数はそれぞれ実際に探査した x の数と、平方数候補として選ばれた x の数である。極端に数が異なるので、前者は百兆を後者は万を単位として表示した。探査数/候補数を表では探/候で表す。単位は百億である。

表 7.3 「FSS」による平方数探査状況

Data	FSS 計算準備				FSS での実行回数		
	素数の数(個)	最大の素数	y 計算回数	y (mod P)回数	探査数(百兆)	候補数(万)	探/候(百億)
A-1	32	137	137	1979	90	17	5
A-2	32	137	137	1881	360	81	4
A-3	32	137	137	1975	1,440	384	4
A-4	32	137	137	1979	5,760	944	6
B-1	32	137	137	2021	180	14	13
B-2	32	137	137	1905	720	86	8
B-3	32	137	137	1905	2,870	311	9
B-4	32	137	137	2021	11,470	721	16

P が 2,3,5,7 の場合は y (mod P)は y (mod P^m)で行った。P^m<65 で m は 1(P=2 では 3)から探査数の減少比が 3 割以上なら 1 増やして決めた。このため、y (mod P)の合計回数は多少変動がある。探/候の欄から、「FSS」では平方数候補の数は

1/数百億に減少していることが分かる。

8. RSA 暗号解読への応用

分解対象数を N とし、FSS を利用して N を素因数分解する方法を検討する。FSS で高速に多倍長整数の平方数が探査できることが判明したので、その利用法を述べる。まだ、検討段階で実用化には今後の改良が必要である。現在検討中の方法は下記に示す 2 方法である。

(1) 2 次式と n 次式による GNFS

(2) FSS を使用した直接因数分解

8.1 2 次式と n 次式による GNFS

RSA 暗号の解読は 1024 ビットや 2048 ビットの合成数 N を $N=P\times Q$ と素因数分解することである。現在 RSA 暗号解読に使用されている GNFS（一般数体ふるい法)は 1 次式と n 次式の組み合わせである。RSA-768(768 ビットの RSA 暗号)の解読[9]には 1 次式と 6 次式が使われた。これを、2 次式と n 次式の組み合わせによる GNFS に置き換える。利点は、GNFS の計算量の 9 割を占めるふるい処理の計算量が削減できることである。一般に、GNFS に使用される二つの多項式を f(x),g(x)としたとき、適当な整数 M を選んで、$f(M)=0 \pmod N$ 及び $g(M)=0 \pmod N$ が成立するように、f(x),g(x)の係数を選ぶ必要がある。これまで、検討されてこなかった理由は $O(N^{1/(n+2)})$ の係数を持つ 2 次式の算出は、2 次式単独でも不可能だと考えられていたためである。FSS を利用すると、2 次式単独の係数を $O(N^{1/(n+3)})$ で算出できる可能性がある。GNFS では、使用する 2 つの多項式の係数の絶対値をできるだけ小さくする。係数の大きさがそのままふるい処理の時間に関係する。1 次式と n 次式の GNFS では両者の係数の絶対値の下限は $O(N^{1/(n+1)})$ である。一方、2 次式と n 次式の GNFS では両者の係数の絶対値の下限は $O(N^{1/(n+2)})$ と小さくできる。1 次式と n 次式の GNFS が現在使用されているのは、1 次式も n 次式も共に $O(N^{1/(n+1)})$ の係数を算出するのが容易なためである。一方、2 次式と n 次式の GNFS は、FSS で 2 次式側の係数が $O(N^{1/(n+2)})$ で求まっても、n 次式側の係数を $O(N^{1/(n+2)})$ まで小さく求めるのは、一般的に不可能と考えられる。そこで、ここでは 2 次式側の係数をより小さくし、n 次式側は容易に求められるよう下記のようにする。

GNFS の 2 次式の係数： $O(N^{1/(n+3)})$

GNFS の n 次式の係数： $O(N^{1/(n+1)})$

d が $O(N^{1/(n+3)})$ 以下で、$M^2=d \pmod N$ となる整数 M を見つけると、これはそのまま $O(N^{1/(n+3)})$ 以下の係数を持つ 2 次式となる。少し変換すれば、$O(N^{1/(n+3)})$ 以下の係数を持つ 2 次式を得るためには、d が $O(N^{2/(n+3)})$ 以下で、$M^2=d \pmod N$ となる整数 M を見つければよい。FSS を使用してこれを見つける方法を以下に示す。

以下の式の係数及び変数は全て多倍長整数とする。

(1) d が $O(N^{2/(n+3)})$ 以下の素数の積を適当に選ぶ（選び方が重要で検討中）。

(2) $c^2=N \pmod d$ で $y=((dx+c)^2-N)/d$ で $|y|$ が平方数になるものを FSS で探す。

(3) $|y|$ は平方数なので $h^2=|y|$ となる h を求める。

(4) $g=|dx+c|, M=gh^{-1} \pmod N$ とすると、$M^2=d \pmod N$ が得られる。

N=778546275812041031941 の例で d は 71 以下の素数とすると、下記が得られる。

d=13　h=1105418538　M=616007064783611099298　$M^2=-13 \pmod N$

d=29　h=3339361173　M=544433284225132756773　$M^2=-29 \pmod N$

d=29　h=3339609195　M=534120873548036143519　$M^2=-29 \pmod N$

d=43　h=877192551　M=748153106548547011035　$M^2= 43 \pmod N$

d=71　h=1640312670　M=1490824621758126757744　$M^2=-71 \pmod N$

この例では d=29 において二つの M で $M^2=-29 \pmod N$ が成立する。その場合、N は直接因数分解される。ただ、大きい N では滅多に二つ M が見つかることは起きない。

8.2 直接因数分解

整数 d に対して、FSS で x を 1 ずつ動かして平方数となる y を複数見つけて、合成数 N を直接因数分解する。ここで、$c^2=N \pmod d$ で $y=((dx+c)^2-N)/d$ である。d=1 は特別で、その時は平方数となる y は一つで良い。「多倍長の平方数の判定は遅い」が定説で、この研究はなされていない。以下にアイデアーを述べるが、まだ効率が悪く、ふるい法に対抗するまでには至っていない。今後の研究に期待する。

d が 1 の場合 c を M とし、N を 4abN で置き換えると $y=(M+x)^2-4abN$ の拡張 Fermat 法になる。ここで、M は N の平方根を整数に切り上げ、x は 0 から順に 1 ずつ増やす。これは、因数分解のデモには向いているが、RSA 暗号解読への応用はほとんど不可能である。可能性は低いが、FSS で京単位の平方数判定情報等を元に、確率的二分法で整数 a,b の値を徐々に決める方法が見つかれば利用できるかも知れない。拡張 Fermat 法は N の因数の比に a,b が超近接できれば、因数分解に利用できる。

拡張 Fermat 法より可能性が高い方法として、下記の方法がある。MPQS（複数多項式ふるい法）で整数 x を動かし、分解対象数 N の平方根より少し小さい d を複数求める。MPQS の A を A^2 にするのが特徴である。

$$((A^2x+b)^2-N)/A^2=a^2d,\ b^2=N \pmod{A^2} \qquad \text{--- (1)}$$

各 d に対し、整数 x を動かして FSS で平方数となる $|y|$ を探す。

$$y = ((dx+c)^2-N)/d,\ c^2=N \pmod{|d|} \qquad \text{--- (2)}$$

同じ d に対し、同じ符号の y で $|y|$ が平方数になるものを二つ見つければ、N は因数分解できる。通常 d は合成数のため c は多数になる。この方法では、(2)式の平方数の一方は(1)式で得られているため、(2)式で求める新しい関係は一つ

で済む。(2)式で得られた二組の c,x を c_1, x_1 と c_2, x_2 で表し、|y|の平方根を h_1, h_2 とすると下記が成立する。

$(dx_1+c_1)^2=dh_1^2 \pmod N$, $(dx_2+c_2)^2=dh_2^2 \pmod N$ --- (3)

(3)式から $S=(dx_1+c_1)h_2$, $T=(dx_2+c_2)h_1$ と置くと、下記が得られる。

$S^2 = T^2 \pmod N$ --- (4)

これにより、$P=S-T \pmod N$, $Q=S+T \pmod N$ とすると N は P と Q に分解される。

以下に異なる素数 A の例を示す。共に N=229910794091155831 とする。例は N が小さい値なので、x の検索範囲も小さく FSS を用いる必要はない。N が大きくなると FSS が必要になる。

(1) A=2731 の例

A^2=7458361, b=1774180 となり、(1)式から下記が得られる。

x=51 で a=15 及び d=-49979238

c は 8 個で、c_1=6444595 と c_2=32295925 から下記が得られる。

x_1=1, h_1=67353 x_2=7, h_2=40965

これから S,T を計算すると下記が得られる。

$25738988755623^2 = 2311402318845^2 \pmod N$

よって、N=455570569×504665599 と分解される。

$A^2x+b=|dx^2+c_2|$=382150591 及び $Aa=h_2$=40965 から二つ目が(1)式に一致。

(2) A=2861 の例

A^2=8185321, b=1209257 となり、(1)式から下記が得られる。

x=-78 で a=9 及び d=265714130

c は 16 個で、$c_1=c_2$=32295925 から下記が得られる。

x_1=2, h_1=25749 x_2=22, h_2=363921

これから S,T を計算すると(1)と同様に

N=455570569×504665599 と分解される。

この場合は一つ目が(1)式に一致する。

9. おわりに

1998 年に無限級数に基づく多倍長精度の計算量を削減する方式 DRM(分割有理数化方式)を考案した。これを実証するため、2002 年に 1.2 兆桁のπ世界記録を達成した。今回、$f(x)=(x+M)^2-N$ などで表される整数多項式 f(x)が平方数になる x を高速に探す方法を考案した。その方法を FSS(高速平方数探査法)と名付けた。FSS を使用すると、2048 ビットの平方数探査で、平方根を計算して判定する方法に比べ、数千万倍高速になることが分かった。現在、FSS を RSA 暗号解読に応用する方法を研究中である。その中で、一番有望なのは、現在の 1 次式と n 次式による GNFS に代わり、2 次式と n 次式の GNFS を使用できるようにすることで

ある。ここで、FSS は今までの方法では不可能とされていた、係数の値が小さい 2 次式の算出に適用が期待される。ただし、この研究は開始したばかりで今後の発展が必要である。

上記 FSS に関する事項以外に、ここでは参考のために、多倍長整数に関する既知の事項を記載した。それは、いずれも多倍長整数に関するもので、その性質、各種乗算手法、級数計算及び現在の暗号と解読法の概要である。読者の役に立てれば幸いである。

＜参考文献＞

1) Torbjorn Graniund and the GMP development team: The GNU Multiple Precision Arithmetic Library, https://gmplib.org/gmp-man-6.1.2.pdf, edition 6.1.2, (2016).
2) 後 保範： 逆数型無限級数の n 桁計算の演算量を削減する前処理方式, 京大数理研研究集会「数値計算における前処理の研究」(1998).
3) 後 保範： 円周率世界記録 6 倍更新の工夫, 早稲田大学 7F セミナ, (2003).
4) 後 保範： 多倍長平方数の高速探査法, 第 45 回数値解析シンポジューム, (2017).
5) 後 保範,金田 康正,高橋 大介： 無限級数に基づく多数桁計算の演算量削減を実現する分割有理数化法, 京大数理研講究録, 1084, 60-71, (1999).
6) 後 保範,金田 康正,高橋 大介： 級数に基づく多数桁計算の演算量削減を実現する分割有理数化法, 情報処理論文誌, Vol.41 No.6, 1811-1819, (2000).
7) 後 保範： 高速剰余変換による多数桁乗算, 情報処理論文誌, Vol.44 No.12, 3131-3138, (2003).
8) 後 保範： 多数桁分割乗算の高速計算法, 情報処理論文誌, Vol.46 No.5, 1266-1273, (2005).
9) 青木 和磨呂: 素因数分解技術の進展; RSA-768 の分解達成への道のり, 情報処理, Vol.51 No.8, 1030-1038, (2010).
10) 松尾 和人: 楕円曲線暗号の基礎２, http://www0.info.kanagawa-u.ac.jp/~matsuo/pub/pdf/chu071006.pdf, (2007).
11) 右田 剛史他３: 級数の集約による多倍長数の計算法とπの計算への応用, 情報処理研究会報告, 98-HPC-74, 31-36, (1998).
12) Hatena Diary: hnw の日記「平方数かどうかを高速に判定する方法」, http://d.hatena.ne.jp/hnw/20140503, (2014).
13) 木田 祐司; 数体ふるい法における因数分解, http://www.rkmath.rikkyo.ac.jp/~kida/nfs_intro.pdf, (2003).