RSA 公開鍵暗号方式の実現

Implementation of RSA Public-key Cryptography

浦 上 浩 一

要 約 インターネットの普及に伴って電子商取引が拡大しつつあるが，この状況下における情報の機密確保，改ざん防止の方法として公開鍵暗号方式が注目を集めている．公開鍵暗号方式の中でもっともよく使用されている RSA 暗号方式を実現するにあたり，次のような工夫，検討を行った．

1） RSA 暗号方式の鍵（かぎ）生成の際に同じ素数を生成しない工夫

ハードデイスクのアクセス時間のばらつきを利用したランダムノイズを導入することにより，疑似乱数発生器に与える種の系列の再現性を著しく低下させた．また，いくつかの実験によってその効果を検証した．

2） 鍵生成の際に必要な素数判定方式の検討

実時間での素数判定が必要なため確率的素数判定方式の二方式を取り上げ比較した．また，ラビン素数判定方式を用いることによって，素数の誤判定確率を実用上問題のないレベルにまで下げることができた．

3） 暗号化および復号処理の高速化

標準的な高速アルゴリズムを用いて，暗号化および復号処理の高速化を行った．

Abstract Over the past few years there has been the growth of the electronic commerce over the Internet. Under this situation the public-key cryptography gives attention to the implementation of technical means in order to keep the confidentiality of messages and to guard against unauthorized modification of messages over a computer network.

This paper discusses some ideas in implementation of RSA cryptography as follows :

1） Inhibiting same prime numbers in generating public keys of RSA cryptography

Introducing random noise acquired based on dispersion of the access time in a hard disk has enormously reduced the recall ratio of seed (bit sting) to be supplied to a pseudo-random number generator. This effect is demonstrated in some of random testing.

2） Primarity testing in public-key generation

This paper compares two probabilistic primarity testing algorithms, to evaluate either of which determines quickly and in real time fashion whether a given positive integer is a prime number. Furthermore, applying Miller-Rabin algorithm has reduced the probability with which a generated random number is not determined as a prime erroneously, to practical level.

3） Quick encryption and/or decryption

A standard high-speed algorithm enables to perform quickly encryption and/or decryption processes.

1. はじめに

インターネットの急激な普及に伴って，電子マネーや電子商取引の実現に対する期待が高まっている．インターネットのような開放されたネットワーク上では，部外者

に対する情報の機密確保,改ざん防止などの必要性が従来に増して重要視されている．そのための手段として暗号，とりわけ公開鍵暗号方式が注目を集めている．

秘密鍵暗号方式であり，米国政府のデータ暗号化標準である DES 暗号[*1] 方式は，1999 年初めに行われた，RSA 社の主催するアルゴリズムの安全性と強度を検証するタイム・クリティカルなコンテスト DES Challenge III において，専用解読ハードウェアによって 22 時間程度で解読された．このことによって強度の高い暗号方式の検討が急務になっている．しかし，公開鍵暗号方式の RSA 暗号[*2] 方式は DES 暗号方式とは使用目的が違うものの，プログラムの変更なく鍵（かぎ）長を長くすれば強度を強化できるという特徴を持っており，コンピュータパワーの進化に合わせて鍵長を安全な長さに増やして行けば安全性を常に確保することができる．

本稿では公開鍵暗号方式の RSA 暗号化エンジンを開発するにあたり，重複のない鍵生成の実現のために数種の工夫を行い，いくつかの実験により工夫の有効性を検証する.また鍵生成の際に必要な確率的素数判定方式の比較を行い実用性の確認を行い，暗号化/復号時の計算の高速化について述べる．

2. RSA 暗号方式

2.1 RSA 暗号方式の原理

まず，RSA 暗号方式の原理について簡単に説明する．RSA 暗号方式では，ある数を法(モジュロ)とする数値変換を行うことにより暗号化を実現している．まず，RSA 暗号方式が利用している数値の興味深い性質を見てみよう．

RSA 暗号方式は，自由に選んだ異なる二つの素数を掛けた数を法とする世界を利用する．そのような世界の例として，二つの素数として 2 と 7 を選び，これらを掛けた数 14 を法とする世界を考える．この世界に存在する全ての数のべき乗を表 1 に示す.

この表より，14 を法とすると全ての数は 7 乗または 13 乗（表 1 の斜字部分）することにより自分自身に戻ることがわかる．このように，二つの素数（それぞれ p，q とする)をかけた数を法とすると，全ての数が自分自身に戻るべき乗数が必ず存在し，このべき乗数は

k を 1 より大きい整数とすると，

$$k \equiv 1 \pmod{(p-1)(q-1)} \quad \cdots\cdots (2\text{-}1)$$

（k は（p−1)(q−1）で割ると 1 余るの意）

を満たす k として求めることができることがわかっている．

先の例では p＝2，q＝7 となり，

$$(p-1)(q-1)=6$$

であるため k は小さい順に 7，13，19，…となり，確かに表 1 で見た通りになることがわかる．

ここでこの原理を暗号に用いるために，k＝ed（e，d ともに 1 より大きい整数）とおけば，まず平文を e 乗することによって暗号文が得られ，次にその暗号文を d 乗することにより平文に戻るので暗号方式として利用できることがわかる．上の場合で

表 1　法を 14 としたべき乗（xk）

x \ べき乗数(k)	1	2	3	4	5	6	*7*	8	9	10	11	12	*13*
1	1	1	1	1	1	1	*1*	1	1	1	1	1	*1*
2	2	4	8	2	4	8	*2*	4	8	2	4	8	*2*
3	3	9	13	11	5	1	*3*	9	13	11	5	1	*3*
4	4	2	8	4	2	8	*4*	2	8	4	2	8	*4*
5	5	11	13	9	3	1	*5*	11	13	9	3	1	*5*
6	6	8	6	8	6	8	*6*	8	6	8	6	8	*6*
7	7	7	7	7	7	7	*7*	7	7	7	7	7	*7*
8	8	8	8	8	8	8	*8*	8	8	8	8	8	*8*
9	9	11	1	9	11	1	*9*	11	1	9	11	1	*9*
10	10	2	6	4	12	8	*10*	2	6	4	12	8	*10*
11	11	9	1	11	9	1	*11*	9	1	11	9	1	*11*
12	12	4	6	2	10	8	*12*	4	6	2	10	8	*12*
13	13	1	13	1	13	1	*13*	1	13	1	13	1	*13*

は式(2-1)を満たすもっとも小さい正整数 e，d は，ed＝k＝25 の場合で

（e，d)＝(5，5)

となる．

表 1 を用いて暗号化，復号の動きを確認すると，次のようにもとに戻ることが確認できる．

	暗号化（e＝5 乗）		復号（d＝5 乗）	
1	→	1	→	1
2	→	4	→	2
3	→	5	→	3
4	→	2	→	4
5	→	3	→	5
6	→	6	→	6
7	→	7	→	7
8	→	8	→	8
9	→	11	→	9
10	→	12	→	10
11	→	9	→	11
12	→	10	→	12
13	→	13	→	13

以上をまとめると公開鍵，秘密鍵，暗号化，復号は以下のようになり，二素数(p，q）の積（n）からなる数の素因数分解の困難性が RSA 暗号方式の安全性を保証している．ただし後述する通り p，q とも数百桁の素数を必要とする．

公開鍵：n，e

秘密鍵：p，q，d

M を平文，C を暗号文とすると，

暗号化：$M^e \equiv C \pmod{n}$ ……(2-2)

復号　：$C^d \equiv M \pmod{n}$ ……(2-3)

2.2 暗号の運用

通常 RSA 暗号方式はトリプル DES[*3] 暗号方式などの共通鍵暗号方式に比べて，計算時間が 1000 倍程度かかると言われている．このため次のように平文はトリプル DES 暗号方式などの共通鍵暗号方式で暗号化し，共通鍵を RSA 暗号方式などの公開鍵暗号方式で暗号化する方式と組合せて使う（図 1）場合が一般的である．

① 平文をトリプル DES 暗号方式の共通鍵を使って暗号化する．

② ①で使用したトリプル DES の暗号鍵（長くても 64 ビット）を RSA 暗号方式の相手の公開鍵で暗号化する．

③ 上で作成した二つを相手に送る．

④ 受信者は自分の RSA 暗号方式の秘密鍵を用いて，②で暗号化した共通鍵を復号する．

⑤復号した共通鍵を用いて平文を復号する．

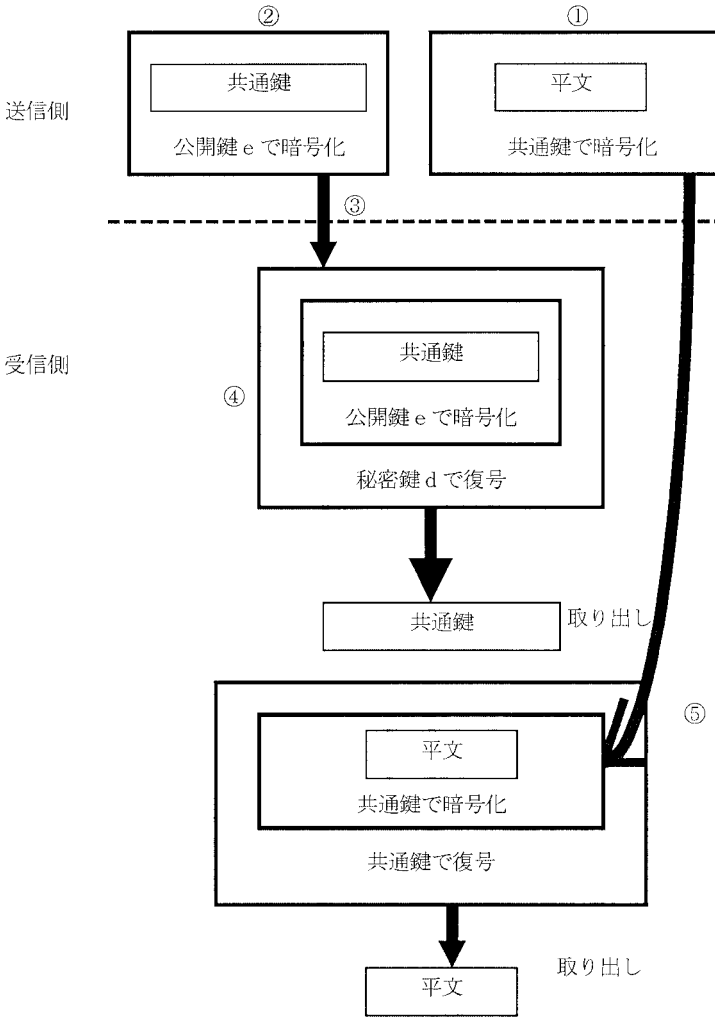

図 1 共通鍵暗号方式と公開鍵暗号方式

2.1 節 RSA 暗号方式の原理の例では 1 桁の素数を用いたが，実際には数百桁の素数が用いられる．現在のコンピュータの計算速度では，n が 1000 ビット程度であれば現在考案されている以上に有効な素因数分解法が見つからない限り安全である．ただ， コンピュータの計算速度の増大に伴って n の桁数も増加させなければならない．現在もっとも有効と考えられる，素因数分解法である数体ふるい法（文献[9]）を用いても 1024 ビットの鍵長では $4*10^{12}$MIPS 年（文献[2]，P. 72）かかり，これ以上有効な素因数分解法が発見されない限り 1999 年現在では十分な鍵長であると言える．さらに 2048 ビットの鍵長であれば後 20 年程度は安全であると見積もられている．

2.3 相互接続性

平文が暗号化の単位となるブロック長に満たない場合にはパディングが必要となる．暗号化時と復号時とでパディング方式が異なるソフトウェアを用いると，相互接続が不可能になる．例えばネットスケープナビゲータ，インターネットエクスプローラに搭載されている SSL（Secure Sockets Layer）は RSA 社の提供する暗号化エンジンを用いており，このエンジンは RSA 社が提唱している Public-Key Cryptography Standards（PKCS）に準拠して開発されている．現状では PKCS が標準であると考え，今回開発した暗号化エンジンの RSA 暗号化/復号時のパディング方式には PKCS を採用し，次のようにした．

ブロックタイプを BT，パディングストリングを PS，データを D とすると，ブロッキング後は次のようになる．‖は文字列の結合演算を表す．

00‖BT‖PS‖00‖D

BT，PS については暗号化の際の鍵によって以下のように設定される．

① 秘密鍵で暗号化する場合
- BT には 1 をセットする．
- PS には 0 xFF を必要数設定する

② 公開鍵で暗号化する場合
- BT には 2 をセットする．
- PS には 0 x 00 でない必要数の乱数を設定する．

3. 素数生成方法

3.1 素数生成処理

次の処理を行い素数の生成を行う．

① 乱数を生成
② 奇数にするために最下位ビットを 1 とする
③ 素数判定を行う．（この方式については 4 章で説明する）
④ 合成数（二つ以上の数の積であらわされる数）と判定された場合は乱数を＋2 し，③へ．

素数と判定されればその数値を素数とする．

3.2 素数の非再現性の実現

3.2.1 素数の非再現性検証の必要条件

素数の非再現性の実現は，同じ鍵（素数 p，q の組）を生成しないための重要な要

件である．仮に同じ鍵が生成され，そのことが判明すれば，機密の確保，情報の改ざんの防止などが不可能になる．鍵を発行する機関が 1 ヶ所のみ存在し，鍵の重複をチェックできるような仕組みが可能でない限り，完全な鍵の重複の回避は不可能である．実際にはあらゆる場所で鍵は生成される可能性があり，鍵の重複の確認ができないため，鍵の完全な重複の回避は不可能である．

今回は実用上問題とならない精度で素数の非再現性を実現する策を考えた．そのためにまず，素数の存在数を考える．十分な数の素数が存在しなければ鍵の非再現性の回避はもともと不可能である．素数判定定理（Prime number theorem）によると N より小さい素数はおよそ N/ln N 個である．よって n ビット長（最上位ビットは常に 1 とする）の素数は，

$2^n/\ln(2^n)/2$

個程度存在することになる．現在（1999 年）では鍵（p または q）長は最低でも 512 ビットは必要とされているので，512 ビット長の素数の数を考えると，

$2^{512}/\ln(2^{512})/2 \quad 1.89\times10^{151}$

個程度存在することになる．この数字から 512 ビット長の素数の存在数は使い切ることのない十分な数であると考えることができる．

次に実用上問題のない素数の非再現性の確認のために，必要条件として次の二つを考えた．

1） 実際に生成されると予想される鍵数を十分に上回る個数の素数を実際に生成し，その素数の非再現性を確認すること．

2） 発生した素数の一様分布性を確認すること．
この一様分布性によってさらに 1）の信頼性を向上させることができる．また解読に対する防御の上でも鍵の分布の偏りはさけなければならない．

3.2.2 実 現 方 法

通常（PGP[*4] など）の対話方式での鍵生成では，ユーザのキータイプ入力の間隔時間を複数サンプリングし，それを種として疑似乱数を発生させる方式がとられることが多い．しかし，ユーザを介さない状況で，再現性のない乱数を発生させるには工夫が必要である．

基本的には，疑似乱数関数が発生する乱数を複数並べて乱数を生成する方法をとった．しかし，単純に時間関数から疑似乱数関数の種を得て，順に疑似乱数関数によって疑似乱数を発生させそれを並べて乱数を生成する方式では，全く同じ時間に起動するなどの原因で種が同じであった場合には，全く同じ乱数を生成してしまうことになる．そこで，疑似乱数関数の種に与える時間値に不規則な影響を与えることによって，乱数生成の再現性をなくす目的で，ハードディスクのアクセス時間のばらつきを利用した．ハードディスクは現在のコンピュータ内には必ず存在し，メモリのアクセス速度に比べて遅い機器であるために，ハードディスクのアクセス時間のばらつきによって生じるアクセス時間の差は，疑似乱数関数の種に用いる時間計測関数の精度で十分測り得る量である．またハードディスクのアクセスはアクセス前のヘッドの位置，ディスクキャッシュの状態，メモリーの状態などにより変わり，この三つの状態はコンピュータの稼働中には刻一刻と変化する．

この効果の確認を行った．ファイルアクセスを行わずに時間計測関数を実行した場合と，ファイルアクセスを行ったあとに時間計測関数を実行した場合の比較である．この実験には PC（UNISYS AQUANTA DM/H（Pentium II 266 MHZ））を用いた．時間計測関数には QueryPerformanceCounter 関数を使用し，この関数を繰り返し実行し，順に関数の返す値の差のばらつきを次の 2 方式で計測した．

方式 1：単純に QueryPerformanceCounter 関数を繰り返し実行し，関数の返す値の差を順に計算する．

方式 2：ファイルアクセス， QueryPerformanceCounter 関数の順に繰り返し実行し，関数の返す値の差を順に計算する．

ただ QueryPerformanceCounter 関数の計測精度が低いため，この関数の返す値の差を連続して計算すると，差に対して精度の影響が大きく出てしまうことがわかった．このため，方式 2 で計算する QueryPerformanceCounter 関数の返す値の差の平均と同程度になるように，方式 1 では QueryPerformanceCounter 関数の返す値の差を計算する間隔を調整した．具体的には 120 回ごとに QueryPerformanceCounter 関数の返す値をサンプリングした．順に関数が返す値の差の平均と標準偏差が表 2 である．方式 1 の平均が 718.5 であるのに対して，方式 2 の平均が 674.1 と，ほぼ同程度に調整され，方式 2 の方が，標準偏差が大きくばらつきが大きいことがわかる．これから疑似乱数発生関数の種に与える時間関数の採取する値に対して，不規則的な効果を与えられることが確認できた．実験に用いた C 言語コードを付録 1 に示す．

表 2　方式別の差の平均と標準偏差

	平均	標準偏差
方式 1	718.5	48.2
方式 2	674.1	283.0

3.2.3　条件 1 の検証

次に前述の条件 1 の非再現性の確認を行った．数十台の PC，UNIX 機を用いて，約 2 ヶ月に渡って素数の生成を行った．その間に生成した約 300 万件の素数の中に重複は存在しなかった．実際には前述の素数 p，q の二つを持つことになるので，両方の鍵が共に重複する確率は $(1/300\text{万})^2=(1/9)^{12}$ 以下ということなる．この数字は「確率的には全世界人口分の数十倍の鍵を生成しても重複は発生しない」という数字である．この低い確率は「重複が仮に発生したとしても，それと気づくことはない」と考えられる確率といえる．

3.2.4　条件 2 の検証

次に条件 2 の一様分布性の確認を行った．条件 1 の実験と同様に数十台の PC，UNIX を用いて，1,196,641 件の素数を生成した．求められた素数（511 ビット）を 16 進表示し頭の 3 桁でグルーピングを行った．グループは 600〜7 FF までの 512 グループあり，グループ内の素数の平均は 2337.2 件，最大は 2472 件，最小は 2197 件，グループ間の素数件数の標準偏差は 46.89 となった．グループごとの件数を図 2 に表

す．これから一様性は十分にあると考えられる．

また χ^2 検定を用いて帰無仮説を「分布は一様である」とすると，自由度 511（グループ数−1）の 5% 限界値 564.70 に対して χ^2 値は 480.81（＜564.70）である．この結果から帰無仮説（「分布は一様である」）を棄却することはできない．

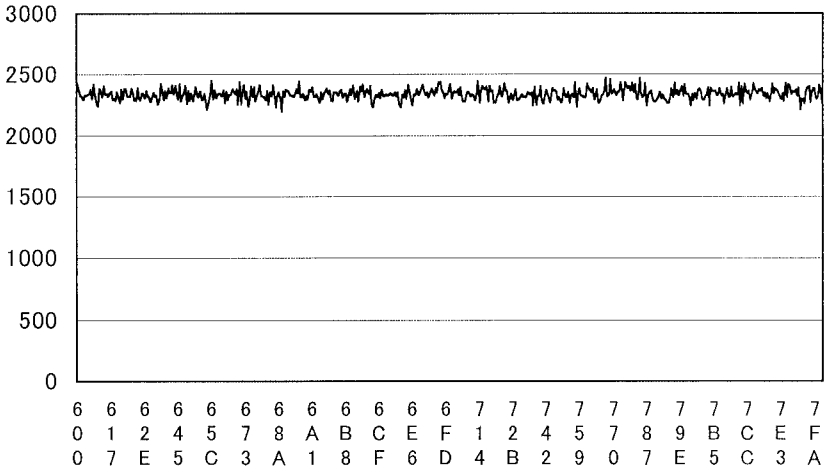

図 2　生成された素数の分布（縦軸：件数，横軸：グループ）

4.　確率的素数判定方法

素数判定に高速化が求められる場合には，確率的素数判定方法が用いられるのが常である．通常は素数候補を 10 進数で 4 桁程度までの素数による割り算を行い，実際に割り切れないことを確かめる高速なテストでふるいにかけたあと，高速なテストに比べて低速な確率的素数判定方法により合成数テストを行うことが多い．素数候補が確率的合成数テストによって合成数と判定されれば，その数は確実に合成数ではある．しかし，逆に合成数と判定されなかった場合でも，誤り確率 u で合成数であると考えることができる．よって，異なるパラメータ（後述）によって複数回合成数テストを繰り返すことによって誤り確率を下げることができる．つまり j 回の合成数テストによって合成数であると判定されなければ，その数は u^j の確率で合成数であると考えることができる．次に具体的な確率的素数判定方法について述べる．

4.1 フェルマー素数判定法

フェルマー素数判定法は誤り確率が 1/2 の判定法である．p が素数であれば p に対して素であるすべての x に対して，

$$x^{**}(p-1) \bmod p = 1 \quad \cdots\cdots (4\text{-}1)$$

が満たされるというフェルマーの定理より，異なる x によって j 回式(4-1)が成立するかどうかのテストを行う．j 種類の x のどれによっても成り立たなければ $(1/2)^j$ の確率で合成数であると見ることができる．ただしフェルマー判定にはカーマイケル数というすべての x で成立する合成数が存在することが知られている．つまりカーマイケル数という合成数がテストされた場合は，決して合成数とは判定されないという欠点をもっている．

4.2　ラビン素数判定法

ラビン素数判定法は誤り確率が 1/4 の判定法である．素数候補 p に対して次のように判定を行う．

① m は奇数で，$p-1=2^k m$ を満たす最小の k を求める

② j 種類の a に対して以下の処理を行い，すべての a に対して「おそらく素数」と判定されれば p が合成数である確率は $(1/4)^j$，一度でも合成数と判定されれば p を合成数と判定する．

③ $b=a^m \bmod$ を計算する．

④ if(b mod p＝＝1)
　then　p は「おそらく素数」

⑤ ④が成り立たない場合
　for i＝0 to k－1（k は 1 の処理で求まる）
　　if(b mod p＝＝－1)
　then　p は「おそらく素数」
　　forloop を抜ける
　else　$b=b^2 \bmod p$
　endfor

⑥ ④，⑤で「おそらく素数」と判定されなければ合成数と判定する．

4.3　両素数判定方式の比較

今回開発した暗号化エンジンでは，ラビン素数判定法を用いて次のように素数判定を行っている．

① 素数テーブル＝{3，…，,9973}の 1237 個の素数のいずれかで割り切れれば合成数と判定するテスト．

② j を 12 としてラビン素数判定法を用いる．

この方式では①のテストを除いて考えても誤り確率は $(1/4)^{12}$ 以下（1677 万分の一以下）となり，実用上十分な精度であると考えられる．

ところで，フェルマー素数判定法は簡単であることから PGP などでも使われている．PGP のソースコードの中には「フェルマー素数判定法で素数と判定された 4,058,000 個の素数は j を 8 としたラビン素数判定法でも素数と判定された」（原文は英語）というコメントがある．そこで j を 4 とするフェルマー素数判定を通ったこの個数以上の素数候補に対して，j を 12 とするラビン素数判定法にかけるテストを行った．結果としては，フェルマー素数判定法によって素数と判定された 1200 万個の素数をラビン判定法は合成数と判定することはなかった．1200 万個のテストが十分であるとは言えないが，この程度の個数では二つの判定法に差は見られなかった．

5.　暗号化，復号の計算の高速化

今回開発した暗号化エンジンでは公開鍵のひとつである e は固定で 17 としている．公開鍵であるため固定とすることで鍵の強度が弱くなることはない．これに対して d の値は n と同程度であり，e に比べて大きい桁になるため式(2-2)で計算される e を用いる公開鍵による暗号化の計算時間の方が，式(2-3)の d を用いる秘密鍵による復

号の計算時間に対して著しく速くなる．計算結果の比較表を表 3 に示す．この計算時間は UNISYS UNIX ワークステーション US 105 による計測結果である．

表 3　公開鍵と秘密鍵による暗号化と復号の計算時間

暗号化／復号	時間（秒）
暗号化	0.17
復号	2.52

100 回の計測結果の平均

5.1　べき乗剰余演算

式(2-2)，式(2-3)の計算には次のようなべき乗剰余演算を行って高速化を図っている．$C=M^e \bmod n$ の場合には，

$$M^e=M\cdot M^2\cdot (M^2)^2\cdot ((M^2)^2)^2\cdot \cdots$$

のように展開できるので，M^2 より $(M^2)^2$ が，$(M^2)^2$ より $((M^2)^2)^2$ が，…と順に計算を行うことができる．また順に mod　n をとることにより桁数を増やすことなく，乗法剰余の計算回数を最大で 2 k（k は e の 2 進表現時のビット数）に減らすことにより，単純に e 乗を計算にその mod n をとる場合に比べて高速に計算できる．

次に暗号化の実例を示す．

秘密鍵 p＝23，q＝19 より公開鍵 n＝437，e＝17，d＝233 とする．

ここで平文 M＝21 とすると，

$$M^e=21^{17}=21\cdot(((212)^2)^2)^2$$

と展開できる．

$$21^2\equiv 4 \pmod{437}$$

となるので，順次求めていくと，

$$(21^2)^2\equiv 4^2 \pmod{437}\equiv 16 \pmod{437}$$

$$((21^2)^2)^2\equiv 16^2 \pmod{437}\equiv 256 \pmod{437}$$

$$(((21^2)^2)^2)^2\equiv 256^2 \pmod{437}\equiv 423 \pmod{437}$$

となるので，

$$M^e\equiv 21\cdot 423 \pmod{437}$$

$$M^e\equiv 143 \pmod{437}$$

暗号文 C は 143 と計算できる．

5.2　中国剰余定理（Chinese Remainder Theorem）

復号の計算時には秘密鍵である p，q を使うことができる．このため前述の方法に加え式(2-3)の計算には中国剰余定理（文献[1]，P. 132）を応用した方法を用いることができる．中国剰余定理は一般化すると次のようになる．

任意の二つの組み合わせが互いに素な正整数を

$$m_1,\ m_2,\ ,\ ,\ ,\ m_N$$

また

$$d_1,\ d_2,\ ,\ ,\ ,\ d_N$$

を正整数とする．

N 個の連立一次合同式

$x \equiv d_i \pmod{m_i}\ (1 \leqq i \leqq N)$

は，

$V = m_1 \times m_2 \times \cdots \times m_N$

を法とする一意な解を持ち，その解は次のようになる．

$$X = \sum_{i=1}^{N} d_i (V/m_i)^{\Phi(m_i)} \bmod V \quad \cdots\cdots(5\text{-}1)$$

ただし，$\Phi()$ はオイラー関数である．

$n = p \cdot q$（p，q ともに素数）であることから式(2-3)は

$M = C^d \bmod (p \cdot q)$

である．中国剰余定理からこの必要十分条件は

$M = C^d \bmod p$ かつ $M = C^d \bmod q$

である．

式(5-1)より

$$M = ((C^d \bmod p) \cdot q^{\Phi(p)} + (C^d \bmod q) \cdot p^{\Phi(q)}) \bmod n$$

ここで p，q は素数であるため，

$\Phi(p) = p - 1$，$\Phi(q) = q - 1$

である．

よって

$$M = ((C^d \bmod p) \cdot q^{p-1} + (C^d \bmod q) \cdot p^{q-1}) \bmod n \quad \cdots\cdots(5\text{-}2)$$

となりこの式を計算すれば M を求めることができる．

乗法剰余の計算回数は

（p の 2 進表現時のビット長＋q の 2 進表現時のビット長）＊2

となるため，前述の方式とほぼ同等であるが，p，q は n のほぼ半分の桁数であるので，剰余の桁数はほぼ半分に減らすことができ高速化が可能である．また $q^{p-1} \pmod n$ や $p^{q-1} \pmod n$ の値は事前に計算しておくことができる．

先程の例から，C＝143，p＝23，q＝19，d＝233 とし，実際に計算してみると次のようになる．

$M = 143^{233} \bmod (23 \cdot 19)$

これは

$M = 143^{233} \bmod 23$ かつ $M = 143^{233} \bmod 19$

となる．

式(5-2)より

$$M = ((143^{233} \bmod 23) \cdot 19^{23-1} + (143^{233} \bmod 19) \cdot 23^{19-1}) \bmod 437$$

を計算すればよい．

前述のべき乗剰余演算を用いてそれぞれ次のように

$143^{233} \bmod 23 = 21$

$143^{233} \bmod 19 = 2$

$19^{22} \bmod 437 = 323$

$23^{18} \bmod 437 = 115$

と計算できる．

これより

$$M=(21\cdot 323+2\cdot 115) \bmod 437$$

よって

$$M=21$$

となり復号できたことが確認できる．

6. おわりに

今回公開鍵暗号方式の RSA 暗号化エンジンを開発した．公開鍵の生成では，一般に用いられているユーザのキータイプ入力の間隔を複数サンプリングし，それを種として必要数の乱数を発生する方法ではなく，ハードディスクのばらつきのあるアクセス時間を利用し，種とて使用する時間関数の返す値にランダムノイズを与えた上で必要数の乱数を発生する方式を採用した．そして，実験によりこの有効性を確認した．また，素数の判定方式については確率的判定を行う二方式を比較すると共に，実験によって実用上問題がないことを確認した．ただ実験で行った限りでは二方式による実際の判定の差は確認できなかった．そして暗号化，復号の計算では一般に用いられている方式を採用し高速化を行った．以上の工夫により実用に十分耐えうる公開鍵暗号方式を実現することができたと考える．

最近では，RSA 暗号方式 1024 ビット相当の安全性強度をわずか 160 ビット程度の鍵長で実現でき，少メモリで済む公開鍵暗号方式のひとつである楕円曲線暗号が世界中で注目を浴びるようになっている．今後 RSA 暗号方式との鍵の安全性，計算速度などの比較も行ってみたい．

＊ 1　Data Encryption Standard の略で，アメリカ商務省標準局が 1977 年に定めた共通鍵暗号方式（文献[1]–P. 76）．

＊ 2　開発者 R. L. Rivest, A. Shamir, L. Adleman の 3 人の頭文字を連ねたもので，1977 年に考案された公開鍵暗号方式．

＊ 3　DES 暗号方式の 56 ビット鍵を二つ使って，3 回暗号化する方式．DES 暗号方式からの移行のしやすさによって DES 暗号方式の後継として使われることが多い．全数検索法によって 22 時間で破られた DES 暗号方式に対して，トリプル DES 暗号方式では全数検索法の解読に 0.9×10^{15} 年程度必要な計算になる．他に解読法もいくつか提案されているが膨大なディスク容量を必要とするなど現時点では現実的ではない．

＊ 4　Pretty Good Privacy の略で電子メール暗号化フリーウェア（文献[3]）．

付録

```
/* ファイルアクセルによるシードへのノイズ添加*/
#include <windows.h>
#include <winbase.h>
#include <stdlib.h>
#include <time.h>
#include <stdio.h>

/* 空ファイルを作る*/
static void GetNoise()
{
  FILE *fpp;

  fpp=
   fopen("DummyFileName.dmy","w");
  if (fpp != NULL) fclose(fpp);
}

__int64 ARRAY[2000];

static void NonNoiseSeed(int id)
{
  LARGE_INTEGER cnt;
  int i;

  for(i= 1; i<=120; i++)
     QueryPerformanceCounter(&cnt);

  ARRAY[id]=  cnt.QuadPart;
}

static void NoiseSeed(int id)
{
  LARGE_INTEGER cnt;
  int i;
  QueryPerformanceCounter(&cnt);
   ARRAY[id] = cnt.QuadPart;
}
```

```
/*ファイルアクセスを行わずにシードを配列にセットする*/
void  NonNoise()
{
      int i;

      for(i = 1; i <= 1000; i++)
      {
            NonNoiseSeed(i);
      }

      for (i = 1; i <= 1000; i++)
        printf("%d  ¥n",ARRAY[i]);
}

/* ファイルアクセスを行ったあとにシードを配列にセットする*/
void  Noise()
{
      int i;

      for(i = 1; i <= 1000; i++)
      {
         GetNoise();
         NoiseSeed(i);
      }

      for (i = 1; i <= 1000; i++)
         printf("%d  ¥n",ARRAY[i]);
}
```

参考文献［1］ Douglas R. Stinson，櫻井幸一監訳「暗号理論の基礎（CryptoGraphy Theory and Practice）」, pp. 123–148.
［2］ 楠田浩二, 櫻井幸一, IMES DISCUSSION PAPER SERIES「公開鍵暗号方式の安全性評価に関する現状と課題」, 日本銀行金融研究所
［3］ Simson Garfinkel 著, 山本和彦監訳,「PGP（暗号メールと電子署名）」,（株）オライリー・ジャパン,
［4］ 岡本龍明・太田和夫共編,「暗号・ゼロ知識証明・数論」, 共立出版
［5］ アロト・サロマー, 足立暁生訳,「公開鍵暗号系」, 東京電気大学出版局
［6］ 辻井重男, 笠原正雄,「暗号と情報セキュリティ」, 昭晃堂
［7］ 松井甲子,「コンピュータのための暗号組立法入門」, 森北出版, pp. 111–116
［8］ 木田祐司, 牧野潔夫,「UBASIC によるコンピュータ整数論」, 日本評論社
［9］ 木田祐司等,「上智大学数学講研録 No. 42「円分数の素因数分解（その 4）」」, 1999/7/16

執筆者紹介 浦 上 浩 一（Koichi Urakami）
1984 年九州大学経済学部経済工学科卒業．同年日本ユニシス（株）入社．エキスパートシステム，ニューラルネット，遺伝的アルゴリズムを適用した金融業，製造業，流通業のユーザーアプリケーションの開発に従事．現在ソリューションシステム部ソリューション技術室に所属，データマイニングなど主に AI 応用システムの開発を中心に暗号エンジン開発，建設 CALS での XML 適用など幅広い分野を担当．