アルゴリズム 30－17
（1992. 11. 21）

高速アルゴリズムを用いたRSA暗号方式

佐竹賢治　　笠原正雄

京都工芸繊維大学 工芸学部 電子情報工学科

京都市左京区松ケ崎御所街道町

あらまし　情報通信技術の発展に伴い、情報ネットワーク上のセキュリティ確保の問題が益々重要視されるようになり、そのため暗号技術が単に情報通信の分野にとどまらず、電子送金、ショッピング、レジャー等の様々な身近な分野で使用されつつある。現在、公開鍵暗号、なかでもRSA暗号が多大の注目を集めているが、暗号化、復号化にべき乗演算及び剰余演算を繰り返し使用するため計算時間が膨大なものになるとう欠点を有している。本稿では法n(2つの素数p、qの積)を特別な形に設定したRSA暗号を提案する。このように法nをある特別な形に設定した場合、安全性が若干犠牲になると考えられるので従来方式と同様の安全性を確保するため、法nを若干大きくしている。本稿ではRSA型暗号の暗号化、復号化の計算時間短縮方法について詳細に述べるとともに、シミュレーション実験によってその効果を確かめている。また、専用ハードウエアを構成するという立場から考察を加えている。

和文キーワード　RSA暗号,　公開鍵暗号,　高速演算

On RSA-type Cryptosystem with Fast Encoding and Decoding Algorithms

Kenji SATAKE　　Masao KASAHARA

Faculty of Engineering and Design Kyoto Institute of Technology

Gosho-kaido-cho Matsugasaki Sakyoku KYOTO 606 JAPAN

Abstract

This paper presents the fast encoding and decoding algorithms using a particular form of divisor n for RSA-type cryptosystem.

Our proposed scheme requires only afew steps for exponentiation modulo n, while the conventional schemes needs almost a hundred steps.

Finally,we have discussed the bounds of caluculation required when our acheme is realized by a hardware.

英文 key words　RSA Cryptopsystem,　Public Key Cryptosystem,　Fast Culclation

1. まえがき

情報通信技術の発展に伴い、情報ネットワーク上のセキュリティ確保の問題が益々重要視されるようになり、そのため暗号技術が単に情報通信の分野にとどまらず、電子送金、ショッピング、レジャー等の様々な身近な分野で使用されつつある。

現在、公開鍵暗号、なかでもRSA暗号が多大の注目を集め、実用化されているが、この暗号は、メッセージの暗号化、復号過程にべき乗演算及び剰余演算を繰り返し実行しなければならないためため暗号化および復号の計算時間が膨大なものになるという欠点を有している。

本稿では法 n を特別な形に設定したRSA暗号を提案する。このように法 n をある特別な形に設定した場合、安全性が若干犠牲になると考えられる。このため、本提案方式では従来方式と同様の安全性を確保するため、法 n を十分に大きくしている。以下ではRSA暗号の暗号化、復号時間の短縮方法について述べる。

2. 基礎的考察

2.1 RSA暗号とその計算時間

RSA暗号は、原理の単純さ、高度の安全性といった優れた特長があるにも関わらず暗号化、復号を行う際に要求される計算量が過大なものになってしまうことが欠点とされている。この理由は暗号化、復号過程においてべき乗剰余演算を繰り返し実行しなければならないことによる。すなわち暗号化では、

$$M^e \equiv C \mod n \tag{1}$$

復号では、

$$C^d \equiv M \mod n \tag{2}$$

なる計算を繰り返し実行しなければならないことによる計算量の増加が問題となる。ただし、

$$\begin{array}{rcl} M & : & \text{メッセージ} \\ C & : & \text{暗号文} \\ e & : & \text{公開鍵} \\ d & : & \text{秘密鍵} \\ n & : & p \cdot q \\ p, q & : & \text{大きな素数 (秘密)} \end{array}$$

である。なお一般にRSA暗号では $2 < e \ll d$ と選ばれることが多いが、本稿では計算時間の短縮化について論ずるので e 及び d にはこのような制約はないとして議論を進める。

2.2 提案方式

本稿では、法 n に特別な形を与えることにより計算時間を短縮した高速演算手法によるRSA型暗号を提案し、検討する。提案方式のシステム構成原理図を図1に示す。

3. 高速演算

3.1 基礎的考察

RSA暗号を使用するに際しては、べき乗剰余演算を高速に行なう専用ハードウェアが必要である。この高速演算に対し加納ら [2] は、シフト加算型乗算を基本とした、剰余テーブル、多入力加算器、高次の拡張Boothアルゴリズムを用いた高速手法を提案している。この手法においては Ci^2 の各ステップにおいての法演算を剰余テーブルと定数加算器を使用することにより高速化している。剰余テーブルのサイズは 2^8 程度ならば実用上十分実現可能となり512ビットの数値メセッージの処理を65ステップで行なうことができる。これに対し本稿では、多大の計算ステップを要するべき乗剰余演算を1ステップで処理することを試みる。

3.2 提案手法

通常 n は $R^x - y$ の形で表される。ここで、この y の桁数に制限を加えれば、法演算が除算を用いずに乗算と加算で置き換えることが可能であり、通常の除算で必要な除数の桁数に比例したステップ数の削減が可能になることが期待できる。以下、このことについて検討しよう。

指数乗算法により M のべき乗を求める場合の第 i ステップにおける値を C_i とする。第 i ステップにおいて n を法とした自乗の演算を遂行する際、第 $i+1$ ステップにおける中間値 C_{i+1} は $C_i^2 \bmod n \longrightarrow C_{i+1}$ として求まる。

ここで、C_i^2 の R^x 以上の部分を $C_i^{(2)}$、R^x 未満の部分を $C_i^{(1)}$ とすると、

$$\begin{array}{rcl} C_i^2 & = & C_i^{(2)} \cdot R^x + C_i^{(1)} \\ & \equiv & C_i^{(2)} \cdot y + C_i^{(1)} \end{array} \tag{3}$$

となる。ここで乗算は並列処理により高速化を行なうものとする。次に C_{i+1}^2 の R^{2x} 以上の部分を $C_{i+1}^{(3)}$、R^{2x}

$$M \rightarrow \boxed{\begin{array}{c} M^e \\ \bmod \ R^x - y \end{array}} \overset{C}{\rightarrow} \boxed{\begin{array}{c} C^d \\ \bmod \ R^x - y \end{array}} \rightarrow M$$

図1. Fast Calculation System

未満 R^x 以上の部分を $C_{i+1}^{(2)}$、R^x 未満の部分を $C_{i+1}^{(1)}$ とすると、

$$\begin{aligned} C_{i+1}^2 &= C_{i+1}^{(3)} \cdot R^{2x} + C_{i+1}^{(2)} \cdot R^x + C_{i+1}^{(1)} \\ &\equiv C_{i+1}^{(3)} \cdot y^2 + C_{i+1}^{(2)} \cdot y + C_{i+1}^{(1)} \end{aligned} \tag{4}$$

となる。そして式 (4) の右辺が n 桁以上の場合でも C_i^2 と同様，これをそのまま次の C_{i+2}^2 の計算ステップで使用する。

通常、$\cdots \longrightarrow C_i^2 \longrightarrow C_{i+1}^2 \longrightarrow C_{i+2}^2 \longrightarrow \cdots$ のステップにおいて桁数は各自乗計算ごとに増加するが $C_{i+1}^{(3)} \cdot y^2$ の桁数が $C_{i+1}^{(2)} \cdot y$ の桁数を 越えないように yの桁数 Y に制限を加えれば，最終のステップでの桁数をある値以下に制限することが可能になる。以下において、このことを検討しよう。

ここで

A	:	n	の桁数
B	:	$C_{i+1}^{(2)} \cdot y$	の桁数
C	:	$C_{i+1}^{(3)} \cdot y^2$	の桁数
D	:	$C_{i+1}^{(3)}$	の桁数
Y	:	y	の桁数

とすると、

$$B = Y + A \tag{5}$$

$$D = 2 \cdot (B+1) - 2 \cdot A \tag{6}$$

$$C = D + 2 \cdot Y \tag{7}$$

である。

$$B \geq C \tag{8}$$

なる条件より

$$Y + A \geq 2 \cdot (Y + A + 1) - 2 \cdot A + 2 \cdot Y \tag{9}$$

である。従って

$$\lfloor \frac{A-2}{3} \rfloor \geq Y \tag{10}$$

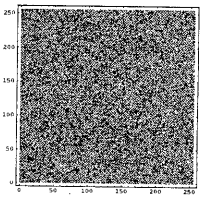
として、最終ステップの桁数を $B+1$ 以下に制限し得る yの桁数の制限範囲が求まる。

なお、yの値は式 (10) を満たす範囲でできる限り大であることが望ましい。さらに、数ステップに 1 回程度の法演算を許容すれば、式 (10) の yの桁に関する制約条件をさらに大きくすることも可能である。

図 2 に R=10 とした場合の画像情報に対する暗号化例を示しておく。なおここで使用した 10 進 180 桁の法 n は、1992 年 8 月 25 日 IT でその素数 p,q を求める因数分解問題をオープンプロブレムとして提示したが現在なお解かれていない。本稿提案方式で使用する法 n が特別の形に限定しているが因数分解に関しては安全性が高いということを示すために提示したものである。

次節において小さな数値例を与えよう。

原画像



暗号化画像

$n =$ 9999999999 9999999999 9999999999 9999999999
9999999999 9999999999 9999999999 9999999999
9999999999 9999999999 9999999999 9999999999
9400474831 7155981399 8058307854 3763306319
5867994492 0788097893

図 2. 画像情報に対する暗号化例 (R=10 の場合)

3.3 高速演算の具体例

yの桁数を上記の範囲に設定すれば、C_iの桁数が高々 $B+1$ 桁以内におさまり、各ステップにおいて C_i^2の演算結果が $2(B+1)$ 桁以下に保たれることを具体例で示す。ここでは説明上 10 進表現を採用することにする。通常の RSA 暗号で行なう指数演算法 (図 3参照) による高速アルゴリズムに従って高速演算の手法を与える。ここで kは e を 2 進展算したときのビット数、e_iは同じく e を 2 進換算したときの下位からの i ビット目の数値を表す。

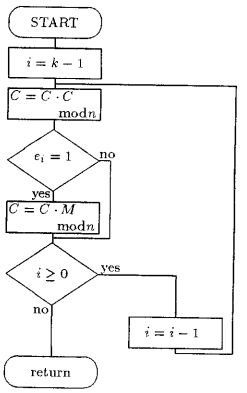


図 3. 高速指数乗算法

今、各パラメータを

$$\begin{aligned} n &= 99999952 \\ e &= 5 = 101_2 \\ M &= 59210376 \\ y &= 48 \\ y^2 &= 2304 \end{aligned}$$

とする。

$k=3,\quad C_0=1$(初期値)

$$
\begin{aligned}
C_1^2 &= C_0 \cdot C_0 = 1 = C_2\\
C_2^2 &= 1 \cdot M = 59210376 = C_3\\
C_3^2 &= C_3 \cdot C_3 = 3505868626061376\\
&= 0 \cdot 2304 + 35058686 \cdot 48\\
&\quad + 26061376\\
&= 1708878304 = C_4\\
C_4^2 &= C_4 \cdot C_4 = 2920265057881916416\\
&= 292 \cdot 2304 + 2650578 \cdot 48\\
&\quad + 81916416\\
&= 209816928 = C_5\\
C_5^2 &= C_4 \cdot M = 12423339198044928\\
&= 1 \cdot 2304 + 24233391 \cdot 48\\
&\quad + 98044928\\
&= 1261250000 = C_6
\end{aligned}
$$

最終ステップが終った段階では、n 以上の値となっている。そこで最終的に、

$$
\begin{aligned}
C_6 \bmod n &\equiv 12\cdot 48 + 61250000 = 61250576\\
&= C
\end{aligned}
$$

として，$M^e \bmod n \equiv C$を求める。ここで、各ステップでの結果が $B+1$ 桁、すなわち高々11 桁以下に保たれていることに注意されたい。

4. 従来方式との比較検討

4.1 従来方式の概要

本稿方式と 3.1で考察した加納ら [2] の提案する高速手法 (以下 KMT 方式と略) を

$$C_i \cdot C_i \bmod n \equiv C_{i+1} \tag{11}$$

なる計算に必要なステップ数 (ここではクロック数の意味合い) で比較するために、KMT 方式の概要について説明する。べき乗剰余演算は図 3から分かるように 式 (11)、すなわち $a\cdot b \bmod n$ なる形の剰余乗算の繰り返しで計算される。KMT 方式ではこれを

$$
\begin{aligned}
S &\equiv a\cdot b \bmod n\\
&\equiv \textstyle\sum_{j=0}^{64}[\; F_3(2^{8j}a \bmod n)\\
&\qquad + F_2(2^{8j}a \bmod n)\\
&\qquad + F_1(2^{8j}a \bmod n)\\
&\qquad + F_0(2^{8j}a \bmod n)\;] \bmod n
\end{aligned}
\tag{12}
$$

と考え、それぞれの部分積をビットシフトのみで構成することにより高速化をはかっている。ここで、F_0〜 F_3 はそれぞれ

$$
\begin{aligned}
F_3 &= \pm 128, \pm 64, 0\\
F_2 &= \pm 32, \pm 16, 0\\
F_1 &= \pm 8, \pm 4, 0\\
F_0 &= \pm 2, \pm 1, 0
\end{aligned}
$$

なる値を取るものとする。$2^{8j}\cdot a \bmod n$ の部分は 2^8個のテーブルルックアップにより $a+\sum_{h=0}^{3} r_h$ なる計算を実行する。また式 (12) の$\sum$の部分を $F\cdot(b_j)$ と書き替えれば実際の$\sum$の部分は、その累算値 Sとの加算により求めることとなり、それを 0 〜 64 までのループ処理を行なうものである。

ここでは、65 回のループにおいて S の範囲を制限しながら計算することが考えられており、S及び $F(b_j)\cdot a$ の値によって定数を加算しこれを実現している。この過程が 65 回繰り返された後、に終段補正として $S<0$ ならば $S=S+$定数 を実行し、その後 $S\equiv S \bmod n$ を行なって $a\cdot b \bmod n$ を求めている。すなわち、

step1 $2^{8j}\cdot a \bmod n$
step2 $S = S + F(b_j)\cdot a$
step3 $S = S + \begin{cases} 768 \\ -768 \\ 0 \end{cases}$
（65 回ループ）
step4 $S = S + 768\cdot n$ $(if\ S<0)$
step5 $S \equiv S \bmod n$
（step1〜step3：剰余乗算部、step4〜step5：終段補正部）

により (11) 式を計算する。従って、剰余乗算部は、3step×65 回の 195 ステップ、終段補正部 2step、計 197 ステップで実現可能である (ブロック図 4参照)。

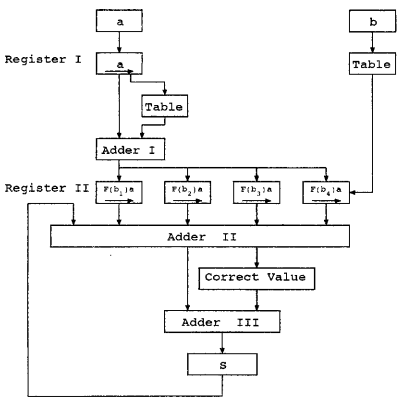


図 4. KMT SCHEME (文献 [2])

4.2 提案方式とのステップ数比較

ここでは、提案方式のステップ数について考える。

step1 $C_i \cdot C_i$
step2 $C_{i+1}^2 \equiv C_{i+1}^{(3)}\cdot y^2 + C_{i+1}^{(2)}\cdot y + C_{i+1}^{(1)}$
（剰余乗算部 (終段補正部なし)）

提案方式では、以上の2ステップで $a \cdot b \equiv \mathrm{mod} n$ の演算が実現できる。ここでは、3.2 でも述べたように、第 i ステップの値を第 $i+1$ ステップにわたす際、n 以下に制限をしていないので、従来方式で見たような終段補正部は不必要になる。しかし、M^e演算を終了した段階の最終段の補正として式 (3) を 1 回だけ実行する必要がある。以上のようにステップ数について比較すると従来方式が 197 ステップであるのに対し、本稿方式では最終段補正を含めたとしても 3 ステップですみ、飛躍的に早い実行速度で、しかも従来通りの安全性を有する RSA 暗号が実現可能であることが明らかである。

5. ハードウェアによる構成

5.1 基礎的考察

従来方式と提案方式の演算時間を比較するため、$R=2$ でのシミュレーションを行なった。ソフトウェアによる多倍長乗算器を用いたため、C_i の桁数が n より増加しただけ、本稿方式では C_i^2の乗算部に時間が必要であった。しかし、他の除算にあたる部分は通常の除算を用いた場合より本提案方式の方がよい結果が得られた。一方、$C_{i+1}^{(3)} \cdot y^2 + C_{i+1}^{(2)} \cdot y + C_{i+1}^{(1)}$ の乗算部分は、y^2、yが $C_{i+1}^{(3)}$ や $C_{i+1}^{(2)}$に比べて小さな桁数であり、また毎ステップ同じ値なので高速乗算アルゴリズムに適し、そのうえ C_i^2 の乗算部分にも高速アルゴリズムが適用可能である。

本提案方式では除算を加算、乗算で構成しており、専用ハードウェアにおいては加算器のみで構成可能である。ここでは、並列処理を用いればさらにどの程度の高速化が可能であるかについて、ハードウェアを構成するという立場から考察する。以下で検討するすべてのハードウェアは HA(Half Adder) と FA(Full Adder) を基本回路としている。これらの加算器は図5のようなゲートのみで構成されたものであるから、加算器入力にデータが入力された時点で加算出力結果が得られる。

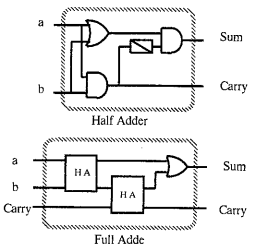


図 5. Adder

5.1.1 R 進加算器の構成

FA を n 個並列に並べて 2 進 n ビット並列加算器を構成する (図 6) 。そしてこの並列加算器の最下位ビットを扱う FA のうち、Carry 入力を使用するものを前段加算器とし、使用しないものを後段加算器とする。これらの前段加算器と後段加算器にさらに R進数の各桁毎の加算結果が Rを越えたかどうかをみるコンパレータと、後段加算器への補正値入力を制御するゲートを用いて、FAM(FULL Adder Module) を構成する。 この R 進並

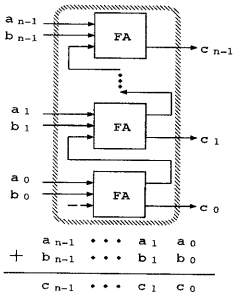


図 6. 2-adic n Bit Adder

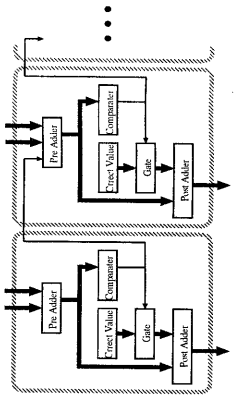


図 7. Full Adder Module

列加算器で桁毎に扱う部分、すなわち 2 進並列加算器での FA 部分に相当する FAM を m 個並べれば図 7のように 2 進化 R進 m 桁の並列加算器が構成できる。

上記の補正値とは、R進数の各桁が Rを越えた場合にその桁の桁上げに対して補正する値である。具体的には

$$2^l - R$$

で与えられる。ただし、lは $2^l > R$を満たす最小値とする。

以上により R進 m 桁の並列加算器は、ゲートのみで構成できることが分かる。このことは、換言すれば R 進加算器の入力部にデータが入力された時点で加算結果出力が得られと言うことを意味しており、クロックを

必要としない通常のゲートICのような振舞いをするように構成できることが明らかである。

5.1.2 R進乗算器の構成

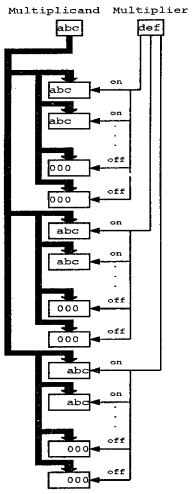


図 8. R-adic n Bit Multiplier

次にR進数の乗算器を3桁同士の乗算といった小さな例でその構成例を示そう(図8)。まず、5ビットのシフトレジスタをR-1個を1組として3ブロック用意する。そしてそれぞれのブロックは、被乗数の入ったレジスタよりハードウェア上で1ビットづつずらして結線されており、1クロックでその被乗数値が各シフトレジスタに入力されるものとする。それぞれのブロックは乗数の各桁毎の値により、その個数分アクティブ状態になるとする。ここでそれぞれのブロックに所属するレジスタを前節のR進加算器を数段設けたものに結線しておく(図8では省略)。このR進加算器は数段重ねた構成になるがすべてゲートのみで実現可能であるため、前節で述べたようにクロックなしでこの加算出力結果が得られる。すなわち、被乗数を各ブロックのレジスタに1クロックで並列入力すればこの時点で乗算結果が出力されることになる。よって、R進乗算器は1クロックのみで出力結果が得られるように構成することが可能である。

5.2 $R^x - y$回路の説明

本提案方式をハードウェア構成した場合のブロック図を図9に示す。この回路は計算で取り扱うパラメータM、e、y、y^2を入れるためのレジスタと、R進乗算器I、II、III、と、R進加算器のみで構成する。この回路の乗算器II、IIIは並列計算可能であり、依存する前段の処理としては乗算器Iの結果のみである。すなわち直列処理を必要とする部分はこの2つの部分のみである。よって、指数演算法の1ステップにあたる$a \cdot b \bmod n$なる計算は、この2つの依存しあう直列処理部分に要するクロック数で計算可能になる。以上のように本提案方式は、ハードウェア構成をすれば、2クロックで演算可能である。これに対しKMT方式(図4)では、

1. レジスタIに被乗数を取り込むクロック、
2. その値の入ったレジスタIをシフトするクロック、
3. そのシフトした値を元にテーブル検索を行なうクロック、
4. レジスタIIに値を取り込むクロック、
5. その値の入ったレジスタIIをシフトするクロック、
6. レジスタIIIに値を取り込むクロック

の計6クロックが必要である(各レジスタのシフトおよびテーブル検索の実行は、実際には1クロックでは完了し得ないので6クロックを大幅に越える値となる)。このKMT方式ではこの6クロックの実行を65ループ繰り返すこととなるので$a \cdot b \bmod n$なる計算に、6×65の390以上のクロック数が実行に費やされることになる。

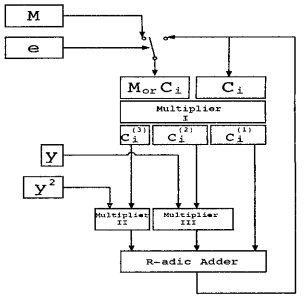


図 9. Fast Calclation

	$a \cdot b \bmod n$ (実行クロック数)	終段補正	最終段補正
KMT方式	390 clock	$S = S + 768 \cdot n$ (if $S < 0$) $S \equiv S \bmod n$	不要
本稿方式	2 clock	不要	$C_i = C_i^{(2)} \cdot y + C_i^{(1)}$ (if $C_i > n$)

6. まとめ

RSA暗号に使用する法nを特別な形に設定すれば計算ステップの削減が大幅に行なえ、ハードウェア構成により従来方式よりも2桁程度の高速化を計れることを示した。具体的には従来方式がα kbpsで暗号通信可能であるとすれば、本稿方式ではほぼ200 α kbps以上の速度で暗号通信可能となることが明らかである。

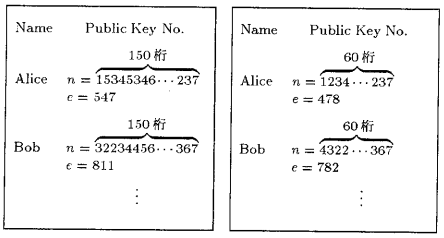
本稿では、安全側に見積り$r^e - s$の形の法が因数分解されやすいとして、桁数を増加しこれを補っている。こ

のためメッセージ長は若干長くなるが、反面、冗長性の高い画像信号を情報源とするような場合には、メッセージの値を大きくとれる方が安全であり、従ってより有利と考えられる。また、公開鍵をいわゆる“電話帳”に10進表現で掲載することを考えれば図10のように従来の方法では法 n を全桁数字掲載する必要があり非常に長いものになるのに対し、本稿方式の共通鍵の場合には連続する9以外のランダムな部分の値のみ掲載すれば良いことになり、60%程度のメモリーの節約になる。

今後の課題として

1. ハードウェア規模の縮小化
2. 32ビットマシーン上での暗号化演算時間の短縮化
3. 応用例(例えば、画像を用いた高速暗号通信)

等が考えられる。



従来方式　　提案方式

図 10. “電話帳”掲載例(R=10の場合)

- 謝辞

本研究を遂行するにあたって著者の一人佐竹は、(株)城南電器工業所 山本勝社長に、貴重な機会とご支援を頂いた。ここに記して深く感謝する。また日頃ご討論頂く笠原研究室の各位に感謝する。

参考文献

[1] 小山謙二：“暗号と素因数分解”,ISEC89-50

[2] 加納康男　松崎なつめ　館林誠：“高次の拡張Boothアルゴリズムを用いたべき乗剰余演算LSI”,1987年暗号と情報とセキュリティ　ワークショップ講演論文集