社団法人 情報処理学会 研究報告
IPSJ SIG Technical Reports

2006－CSEC－34 (13)
2006／7／20

コプロセッサの2倍のビット長をもつモンゴメリ乗算

吉野 雅之† 桶屋 勝幸† ヴィオムカミーユ†

† (株) 日立製作所 システム開発研究所 〒215-0013 川崎市麻生区王禅寺 1099 番地
E-mail: †{m-yoshi,ka-okeya,camille}@sdl.hitachi.co.jp

あらまし 本稿では, n ビットのモンゴメリ乗算を実行するコプロセッサを用い, $2n$ ビットのモンゴメリ乗算を計算する手法を提案する. RSA 暗号を代表とする公開鍵暗号系は, 安全性を確保するため, 年々より長いビット長を要求している. その一方, IC カード等が暗号処理に利用するコプロセッサのビット長は固定であり, 一部のコプロセッサは増加する RSA 暗号のビット長に対応できない. 本稿では, コプロセッサに変更を加えずに, コプロセッサのビット長以上の RSA 暗号を実現する計算手法を提案する. 古典的な剰余演算に基づく他のビット長 2 倍化技術と異なり, 本稿で提案する手法は, 多くのコプロセッサが採用するモンゴメリ乗算に着目し, その計算の高速性を生かすように設計した.
キーワード モンゴメリ乗算, ビット長 2 倍化技術, RSA 暗号, コプロセッサ, IC カード

Double-Size Montgomery Multiplication of a Crypto-Coprocessor

Masayuki YOSHINO †, Katsuyuki OKEYA †, and Camille VUILLAUME †

† Hitachi, Ltd., Systems Development Laboratory, Kawasaki, Japan.
E-mail: †{m-yoshi,ka-okeya,camille}@sdl.hitachi.co.jp

Abstract We present a novel approach for computing $2n$-bit Montgomery multiplications with n-bit hardware Montgomery multipliers. Smartcards are usually equipped with such hardware Montgomery multipliers; however, due to progresses in factoring algorithms, the recommended bit length of public-key schemes such as RSA is steadily increasing, making the hardware quickly obsolete. Thanks to our double-size technique, one can re-use the existing hardware while keeping pace with the latest security requirements. Unlike the other double-size techniques which rely on *classical* n-bit modular multipliers, our idea is tailored to take advantage of n-bit *Montgomery* multipliers. Thus, our technique increases the perenniality of existing products without compromises in terms of security.
Key words Montgomery multiplication, double-size technique, RSA, crypto-coprocessor, smartcard

1. はじめに

Montgomery が提案した "モンゴメリ乗算" [7] は, RSA 暗号などの公開鍵暗号系が広く実装する, 剰余演算の計算手法である [14]. 他の計算手法と異なり, モンゴメリ乗算はキャリーによる遅延の影響を受けない高速な剰余演算であり, 特にハードウェア実装に適する. IC カードなどの小型機器では, コプロセッサと呼ばれる演算器にモンゴメリ乗算をハードウェア実装し, データの暗号化／復号化や電子署名の生成／検証などの処理時間を短縮している. 一方, 近年の素因数分解技術の発展から [13], セキュリティ関連機関は, RSA 暗号の推奨ビット長を, 現在主流の 1024 ビットから, 2048 ビットへ移行し始めている. しかし, ハードウェア実装であるコプロセッサは処理できるビット長に上限があり [9], その多くが 2048 ビットに対応できない.

将来的に多くのコプロセッサのビット長が不足するため, 現状のコプロセッサを用い, そのビット長の 2 倍の剰余演算を計算する手法が盛んに研究されている. 秘密鍵を用いる場合 (電子署名の生成やデータの復号化) では, 中国人剰余定理 [6] を用い, コプロセッサのビット長の 2 倍の剰余演算を実現する手法が提案された [11]. しかし, 公開鍵を用いる場合 (電子署名の検証やデータの暗号化) には, 上記の中国人剰余定理が利用できないため, コプロセッサより大きいビット長の剰余演算は計算できなかった.

Paillier は, 剰余演算のビット長 2 倍化手法の研究の先駆者であり, n ビットの古典的な (モンゴメリ乗算を実装しない) コプロセッサと公開鍵を用い, kn ビットの古典的な剰余演算を効率的に計算する手法を示した [9]. 次いで, Fischer らは, $2n$ ビットの場合に特化し, Paillier の手法を改良した [3]. さらに, Chevallier-Mames らが, Fischer らと同様に $2n$ ビットの場合において, 汎用的なモジュラスと特殊なモジュラスを対象に, より最適化した手法を示した [2]. これらの研究により, 一部のコプロセッサはビット長の制約から開放されたが, 残念ながら, 適

用対象でないコプロセッサが存在する．特に，多くのICカード上のコプロセッサは，剰余演算の高速手法である，モンゴメリ乗算を実装している．モンゴメリ乗算は，古典的な剰余演算とは演算の性質が異なり，モジュラスに制約条件を課す．古典的な剰余演算を対象とする彼らの提案手法はこの条件を満たしておらず，従って，モンゴメリ乗算を実装するコプロセッサには，ビット長の厳しい制約がいまだに課されている．

本稿では，n ビットのモンゴメリ乗算を実装するコプロセッサを用い，$2n$ ビットのモンゴメリ乗算を計算する手法を提案する(以降では，区別のため，モンゴメリ乗算を実装するコプロセッサをモンゴメリ乗算器と呼び，古典的な剰余演算を実装する場合は古典的な剰余演算器と呼ぶ)．まず，$2n$ ビットのモンゴメリ乗算の計算に必要となる，モンゴメリ乗算の商を定義する．商の計算には二種類の実行環境を検討し，それぞれ効率的な計算手法を示す．一つ目の実行環境は，モンゴメリ乗算器は全く変更できず，乗算器の再利用，すなわちソフトウェア実装を想定する．この環境において，n ビットのモンゴメリ乗算器を2回呼び出し，擬似的に商を計算するアルゴリズムを示す．二つ目の実行環境は，ハードウェアであるモンゴメリ乗算器は修正可能であるものの，そのビット長が n に制限される場合のハードウェア実装を想定する．このとき，商の計算に必要な，モンゴメリ乗算器のアルゴリズムの最小限の変更を示す．次いで，n ビットのモンゴメリ乗算器用に，入力された $2n$ ビットの整数に対する新しい整数表現を提案する．モンゴメリ乗算に設定するモジュラスは常に奇数でなければならない．新しい整数表現は，モジュラスを常に奇数に設定しつつ，$2n$ ビットの整数を通常のバイナリ表現から効率的に変換または逆変換する．

結論として，本稿で提案するビット長2倍化技術は，n ビットのモンゴメリ乗算器を用い，$2n$ ビットのモンゴメリ乗算を実現する．従って，現世代の多くのコプロセッサが，より高い安全性と長いビット長を要求する次世代にも活躍できる．

本稿の構成は次の通りである．2. 章では，古典的な剰余演算器に基づく既存のビット長2倍化技術[2], [3]を説明する．3. 章では，モンゴメリ乗算を紹介した後，モンゴメリ乗算における既存のビット長2倍化技術の問題点を示す．4. 章では，本稿の提案手法である，n ビットのモンゴメリ乗算器を用いた $2n$ ビットのモンゴメリ乗算を説明する．本稿の提案手法では，n ビットのモンゴメリ乗算の商が必要である．5. 章では，二通りの実行環境において，モンゴメリ乗算の商の計算手法を示す．一方はソフトウェア実装に適しており，もう一方はハードウェア実装に適する．6. 章で，実験による検証結果と，実装時に生じる問題点を示し，最後に 7. 章で結論を述べる．

表記法

本稿では，英字 n はモンゴメリ乗算器または古典的な剰余演算器のビット長を表す．また，大文字で表された A, B, N, M などの英字は $2n$ ビットの整数とし，小文字で表す英字は他のビット長 (主に n ビット) の整数を表す．

2. 従来研究

ICカードなどの小型機器では，コプロセッサと呼ばれる演算器を搭載し，公開鍵暗号系の主要な演算である剰余演算を高速に処理する．しかし，ハードウェア実装であるコプロセッサは，処理できるビット長に上限があり，公開鍵暗号系の実装に適さない場合がある．特に，近年の素因数分解技術の進歩[13]から，主要なセキュリティ機関は，公開鍵暗号系の代表格である RSA 暗号に対し，近い将来，1024 ビットから 2048 ビットへ移行するよう推奨している．ビット長の上限が 2048 ビット未満のコプロセッサは 2048 ビットの RSA 暗号を実行できないため，この問題を一般化した，n ビットのコプロセッサを用いて $2n$ ビットの剰余演算を計算する，多くの研究を生んだ．

Paillier は $2n$ ビットの剰余演算の研究の先駆者であり，n ビットの古典的な剰余演算器と公開鍵を用い，kn ビットの古典的な剰余演算を効率的に計算する手法を示した．次いで，Fischer らは，$2n$ ビットの場合に特化し，Paillier の手法を改良した[3]．さらに，Chevallier-Mames らは，同様に $2n$ ビットの場合に取り組み，より最適化した $2n$ ビットの古典的な剰余演算の計算手法を示した[2]．本章では，Chevallier-Mames らの手法ではなく，提案手法が利用した Fischer らの手法を紹介する．

2.1 Fischer らの提案手法

本節では，Fischer ら[3]が提案した，古典的な剰余演算器を用い，その2倍のビット長をもつ古典的な剰余演算の計算手法を示す．彼らの手法では，その計算過程で，n ビットの剰余演算の剰余のほかに，n ビットの商を必要とする．以下に，整数 x と y の積，モジュラス w，商 q_c，剰余 r_c の関係を方程式で表す．

$$xy - q_c w = r_c$$

古典的な剰余演算の基本アイデアは，積 xy がモジュラス w より小さくなるよう，積 xy からモジュラス w を q_c 回分，減算を繰り返すことである．

Fischer らは，古典的な剰余演算に基づく n ビットの商と剰余を計算する MultModDiv 命令を仮定した．ここで，n ビットの正の整数 x, y, w に対し，MultModDiv 命令は以下の方程式を満たす．

$$\mathsf{MultModDiv}(x, y, w) = (q_c,\ r_c)$$

ただし $q_c = \lfloor (xy)/w \rfloor$, $r_c = xy \pmod{w}$ である．

MultModDiv 命令は，古典的な剰余演算を拡張した計算である．古典的な剰余演算を実装する剰余演算器が利用できる場合，この剰余演算器を呼び出し，または剰余演算器の回路を一部変更し，MultModDiv 命令を擬似的または直接的に実行できる．

Fischer らが提案した，$2n$ ビットの古典的な剰余演算を計算するアルゴリズムを紹介する．まず，不等式 $A, B < N$ を満たす $2n$ ビットの整数 A, B, N を与える．MultModDiv 命令のビット長にあわせ，次に，$2n$ ビットの整数 A, B, N を，以下の方程式に従い，n ビットの整数に分割する．

$$A = a_1 2^n + a_0,\ B = b_1 2^n + b_0,\ N = n_1 2^n + n_0$$

上記の変換により成立する方程式 $n_1 2^n \equiv -n_0 \pmod{N}$ から，

Fischer らは, $2n$ ビットの剰余演算を計算するアルゴリズム 1 を提案した.

アルゴリズム 1: Fischer らのアルゴリズム

入力: $A = a_1 2^n + a_0$, $B = b_1 2^n + b_0$, $N = n_1 2^n + n_0$ ただし $0 \leqq A, B < N$, $2^{2n-1} < N < 2^{2n}$.
出力: $AB \pmod N$.

1. $(q_1, r_1) = \mathsf{MultModDiv}(b_1, 2^n, n_1)$
2. $(q_2, r_2) = \mathsf{MultModDiv}(q_1, n_0, 2^n)$
3. $(q_3, r_3) = \mathsf{MultModDiv}(a_1, r_1 - q_2 + b_0, n_1)$
4. $(q_4, r_4) = \mathsf{MultModDiv}(a_0, b_1, n_1)$
5. $(q_5, r_5) = \mathsf{MultModDiv}(q_3 + q_4, n_0, 2^n)$
6. $(q_6, r_6) = \mathsf{MultModDiv}(a_1, r_2, 2^n)$
7. $(q_7, r_7) = \mathsf{MultModDiv}(a_0, b_0, 2^n)$
8. **Return** $(r_3 + r_4 - q_5 - q_6 + q_7)2^n + (r_7 - r_6 - r_5)$

3. モンゴメリ乗算

Montgomery が提案したモンゴメリ乗算 [7] は, RSA 暗号などの公開鍵暗号系が広く実利用する, 剰余演算を高速に処理する計算手法である. モンゴメリ乗算はハードウェア実装に適しており, 例えば, IC カードなどの小型機器では, モンゴメリ乗算をハードウェア実装したコプロセッサを用い, データの暗号化／復号化, 電子署名の生成／検証などを短時間で処理している. 本章では, モンゴメリ乗算を紹介した後, 従来のビット長 2 倍化技術をモンゴメリ乗算器に適用する際の問題を説明する.

3.1 モンゴメリ乗算のアルゴリズム

$0 \leqq x, y < w$ かつ $\gcd(w, m) = 1$ を満たす n ビットの整数 x, y, w に対し, モンゴメリ乗算は以下の方程式を満たす剰余 r を出力する.

$$r := xym^{-1} \pmod w$$

ただし, m はモンゴメリ定数と呼ばれる定数である. モンゴメリ定数の値として, $m = 2^n$ が広く実利用されている. $m = 2^n$ の場合, 演算中にシフトやメモリ操作などの基本操作を広く使用でき, さらに, キャリーの発生による遅延の影響も受けなくてすむ [6]. モンゴメリ乗算の基本的なアイデアは, 計算コストの高い除算を低い乗算と加算へ置換することである (図 1 に原理を示す.). 最上位ビットから始める古典的な剰余演算とは異なり, モンゴメリ乗算では, 最下位ビットから (即ち右側から左側に) モジュラス w を積 xy に足して行き, 最上位ビット側に剰余 r を保存する.

3.2 従来技術における問題点

Fischer ら [3] と Chevallier-Mames ら [2] は, n ビットの古典的な剰余演算を用い, $2n$ ビットの古典的な剰余演算の計算手法を提案した. しかし, 彼らの手法をモンゴメリ乗算器に適用するには, 以下の問題がある.

問題 1 : 商.

従来のビット長 2 倍化技術では, $2n$ ビットの剰余を計算するため, n ビットの剰余に加え, n ビットの商も必要である. しかし, モンゴメリ乗算には商そのものが定義されていない.

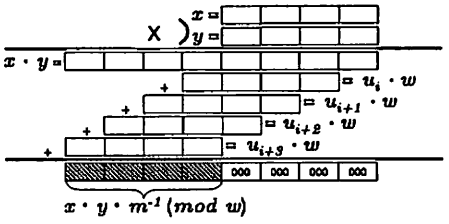

図 1 モンゴメリ乗算の基本的な原理

問題 2: モジュラス.

モンゴメリ乗算において, モジュラスはモンゴメリ定数と互いに素でなければならない. 従って, モンゴメリ乗算におけるモジュラスは必ず奇数である. しかし, 彼らの手法では, 偶数のモジュラスを必要とする場合がある.

4. 提案方式

本章では, モンゴメリ乗算を実装する n ビットの剰余演算器を用いた, $2n$ ビットのモンゴメリ乗算の計算方式を提案する. モンゴメリ乗算は, データの暗号化／復号化, 電子署名の生成／検証などの公開鍵暗号系の処理を高速化し, ハードウェア実装されることが多い. その一方で, モンゴメリ乗算器に適用できる, モンゴメリ乗算のビット長 2 倍化技術が存在しない.

4.1 剰余を求める命令

定義 1 において, モンゴメリ乗算の剰余を計算する MultMon 命令を定義する.

定義 1. $0 \leqq x, y < \min\{w, 2^n\}$, $2^{n-1} < w < 2^{n+1}$, $m = 2^n$, $\gcd(m, w) = 1$ を満たす整数 x, y, w, m が与えられたとき, MultMon 命令を次の方程式で定義する.

$$r = \mathsf{MultMon}(x, y, w)$$

ただし, $r := xym^{-1} \pmod w$ である.

4.2 商を求める命令

$2n$ ビットの古典的な剰余演算を実現するため, Fischer ら [3] と Chevallier-Mames ら [2] の提案手法は n ビットの古典的な剰余演算の商を用いた. 提案する $2n$ ビットのモンゴメリ乗算の計算手法も n ビットのモンゴメリ乗算の商を必要とする. しかし, モンゴメリ乗算は, 剰余の計算に特化した手法であり, 商自体が定義されていない. そこで, 商の概念をモンゴメリ乗算へ拡張し, モンゴメリ乗算における商を新たに定義した.

モンゴメリ乗算が計算する剰余は, 古典的な剰余演算が計算する剰余とは, 異なる値をもつ. 実際, 定義 1 から, 次の方程式が成り立つ.

$$xy \equiv rm \pmod w,$$

ただし, $m = 2^n$ とする. 上記の式は, 次の方程式

$$xy - qw = rm$$

を満たす整数 q の存在を意味する．この整数 q を，以降では，モンゴメリ乗算の商と呼ぶ．

モンゴメリ乗算の商と剰余を計算する MultMonDiv 命令を定義する．

定義 2. $0 \leqq x, y < \min\{w, 2^n\}$, $2^{n-1} < w < 2^{n+1}$, $m = 2^n$, $\gcd(m, w) = 1$ を満たす整数 x, y, w, m に対し，MultMonDiv 命令を次の方程式で定義する．

$$(q, r) = \mathsf{MultMonDiv}(x, y, w)$$

ただし $r := xym^{-1} \pmod w$ かつ q は方程式 $xy = qw + rm$ を満たす整数である．

MultMonDiv 命令を実行するアルゴリズムは 5. 章にて詳細に示す．

4.3 $2n$ ビットの整数表現

従来のビット長 2 倍化技術と同様，提案手法も，n ビットのモンゴメリ乗算器が処理できるよう，$2n$ ビットの整数を n ビットの整数に分割する．$2n$ ビットの整数を分割する，整数表現を定義 3 に示す．

定義 3. $0 \leqq A, B < N$, $2^{2n-1} < N < 2^{2n}$ を満たす $2n$ ビットの整数 A, B, N を次の方程式で表現する．

$$N = n_1 z + n_0 m,\ A = a_1 z + a_0 m,\ B = b_1 z + b_0 m.$$

ただし，整数 m, z は方程式 $2^{n-1} < z < 2^n$, $m = 2^n$, $\gcd(z, m) = 1$ を満たす．

定義 3 の整数表現に従うと，N が奇数である場合，積 $n_0 m$ は必ず偶数であるので，n_1 と z は必ず奇数である．

4.4 修正した Fischer らのアルゴリズム

定義 3 の整数表現を用い，Fischer らのアルゴリズムをモンゴメリ乗算へと拡張する．アルゴリズム 2 は，奇数である n_1 と z のみをモジュラスとして用い，モンゴメリ乗算におけるモジュラスの問題を解決する．モンゴメリ乗算器は n ビットの剰余 $(xym^{-1} \pmod w$ ただし $0 \leqq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n)$ を出力するので，提案するアルゴリズム 2 は $2n$ ビットのモンゴメリ乗算の剰余 $(ABM^{-1} \pmod N$ ただし $0 \leqq A, B < N$, $2^{2n-1} < N < 2^{2n}$, $M = 2^{2n})$ を出力する．

アルゴリズム 2: 修正した Fischer らのアルゴリズム

入力: $A = a_1 z + a_0 m$, $B = b_1 z + b_0 m$, $N = n_1 z + n_0 m$, ただし $M = 2^{2n}$.
出力: $ABM^{-1} \pmod N$.

1. $(q_1, r_1) = \mathsf{MultMonDiv}(b_1, 2^n, n_1)$
2. $(q_2, r_2) = \mathsf{MultMonDiv}(q_1, n_0, z)$
3. $(q_3, r_3) = \mathsf{MultMonDiv}(a_1, r_1 - q_2 + b_0, n_1)$
4. $(q_4, r_4) = \mathsf{MultMonDiv}(a_0, b_1, n_1)$
5. $(q_5, r_5) = \mathsf{MultMonDiv}(q_3 + q_4, n_0, z)$
6. $(q_6, r_6) = \mathsf{MultMonDiv}(a_1, r_2, z)$
7. $(q_7, r_7) = \mathsf{MultMonDiv}(a_0, b_0, z)$
8. **Return** $(r_3 + r_4 - q_5 - q_6 + q_7)z + (r_7 - r_6 - r_5)m$

定理 1: $0 \leqq A, B < N < 2^{2n}$ かつ $M = 2^{2n}$ を満たす $2n$ ビットの整数 A, B, M, N が与えられたとき，アルゴリズム 2 は，MultMonDiv 命令を呼びだし，$2n$ ビットのモンゴメリ乗算の剰余 $ABM^{-1} \pmod N$ を計算する．

証明． 定理 1 の証明は，付録 1. に示す． □

n ビットの古典的な剰余演算器を用いた $2n$ ビットの古典的な剰余演算の計算手法において，Chevallier-Mames らが提案する手法は，Fischer らの手法を改良し，剰余演算器の呼出回数が 1 回少ない．しかし，Chevallier-Mames らの手法では，モジュラスに z と $z-1$ を用いているため，必ずいずれかが偶数となる．そこで，本稿では，Chevallier-Mames らの手法ではなく，Fischer らの手法をモンゴメリ乗算に拡張した．

5. 商の計算

4. 章において，$2n$ ビットのモンゴメリ乗算の計算に必要となる，モンゴメリ乗算の商を新たに定義した．本章では，二種類の実行環境を検討し，それぞれの場合において，モンゴメリ乗算の商を効率的に計算するアルゴリズムを示す．一つ目の実行環境では，回路が変更不可能なモンゴメリ乗算器を想定し，これを再利用するアルゴリズムを設計した．この場合，提案するビット長 2 倍化手法は，純粋なソフトウェアのみの実装を可能とする．5.1 節において，モンゴメリ乗算器を 2 回呼び出し，モンゴメリ乗算の商を擬似的に計算するアルゴリズムを示す．二つ目の実行環境では，モンゴメリ乗算器の回路は一部変更可能であるが，最大ビット長は n に限定されている場合を想定する．5.2 節において，モンゴメリ乗算器の変更を最小限に抑えるアルゴリズムを示す．

5.1 ソフトウェアによるアプローチ

モンゴメリ乗算器を 2 回呼び出し，n ビットのモンゴメリ乗算の商を計算するアルゴリズム 3 を示す．

アルゴリズム 3: MultMon 命令に基づく MultMonDiv 命令

入力: x, y, w ただし $0 \leqq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$, $\gcd(w, m) = 1$.
出力: q, r.

1. $r \leftarrow \mathsf{MultMon}(x, y, w)$
2. $r' \leftarrow \mathsf{MultMon}(x, y, w + 2^n)$
3. $tmp \leftarrow xy - rw + r'(w + 2^n) \pmod{2^2}$
4. **If** $tmp > 2$, **then** $tmp \leftarrow tmp - 2^2$.
5. $q \leftarrow tmp \times (w + 2^n) + r' - r$
6. **Return** (q, r)

アルゴリズム 3 の正当性を示すため，補題 1 から補題 3 を示す．補題 1 では，モンゴメリ乗算の商の値域について述べる．

補題 1. $0 \leqq x, y < \min\{w, 2^n\}$, $2^{n-1} < w < 2^{n+1}$, $m = 2^n$ を満たす整数 x, y, w, m が与えられ，かつ $\mathsf{MultMon}(x, y, w)$ より整数 r が得られる場合，方程式 $xy = qw + rm$ を満たす整数 q に対し，$0 \leqq q < 2^{n+1}$ が成り立つ．

証明. $0 \leqq xy - rm < 2^{2n}$ より, $0 \leqq qw < 2^{2n}$ が成り立つ. $w > 2^{n-1}$ から, $0 \leqq q < 2^{n+1}$ が成り立つ. □

補題 2 は, 異なる 2 つのモンゴメリ乗算の商の値域の差分について述べる.

補題 2. $0 \leqq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$, $\gcd(w, m) = 1$ を満たす整数 x, y, w, m が与えられ, かつ MultMon(x, y, w) より整数 r, MultMon$(x, y, w+2^n)$ より整数 r' が得られ, 整数 q が方程式 $xy = qw + rm$, 整数 q' が方程式 $xy = q'(w+2^n) + r'm$, 整数 δ が方程式 $\delta = xy + rw - r'(w+2^n) \pmod{2^2}$ を満たすとする. このとき, 方程式 $q' - q = \delta 2^n$ または $q' - q = (\delta - 2^2)2^n$ が成り立つ.

証明. $(w+2^n)^{(-1)} - w^{(-1)} = 2^n \pmod{2^{n+2}}$ より, 以下の式が成り立つ.

$$\begin{aligned} q' - q &= xy((w+2^n)^{-1} - w^{-1}) \\ &\quad - r'm(w+2^n)^{-1} + rmw^{-1} \\ &\equiv xy2^n - r'(w+2^n)^{-1}m + rw^{-1}m \pmod{2^{n+2}} \\ &= \{xy - r'(w+2^n)^{-1} + rw^{-1}\} \times 2^n \end{aligned}$$

$(w+2^n)^{-1} = w^{-1} = w \pmod{2^2}$, より, 次の方程式を得る.

$$\begin{aligned} q' - q &\equiv \{xy - r'(w+2^n) + rw \pmod{2^2}\} \times 2^n \pmod{2^{n+2}} \\ &= \delta \times 2^n \end{aligned}$$

補題 1 より $-2^{n+1} < q' - q < 2^{n+1}$ が導けるので, $-2 < \delta < 2$ が成り立つ. 従って, $\delta < 2 \pmod{2^2}$ である場合には, 方程式

$$q' - q = \delta \times 2^n$$

が成り立つ. それ以外のときには, 以下

$$q' - q = (\delta - 2^2) \times 2^n$$

が成り立つ. □

補題 3 は, モンゴメリ乗算の商を計算する方程式を述べる.

補題 3. $0 \leqq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$, $\gcd(w, m) = 1$ を満たす整数 x, y, w, m が与えられ, かつ MultMon(x, y, w) より整数 r が, MultMon$(x, y, w+2^n)$ より整数 r' が得られ, かつ整数 q が方程式 $xy = qw + rm$, 整数 q' が方程式 $xy = q'(w+2^n) + r'm$ を満たし, 整数 δ が方程式 $\delta = xy + rw - r'(w+2^n) \pmod{2^2}$ を満たすとする. このとき, 方程式 $q = \delta(w+2^n) + r' - r$, または $q = (\delta - 2^2)(w+2^n) + r' - r$ が成り立つ.

証明. $xy = qw + rm = q'(w+2^n) + r'm$ より, 方程式 $q2^n = (q - q')(w+2^n) + (r - r')m$ を得る. 補題 2 より, $q = \delta \times 2^n$ または $q = (\delta - 2^2) \times 2^n$ が成り立つので, 以下の方程式を得る.

$$q = \delta(w+2^n) + r' - r,$$

または

$$q = (\delta - 2^2)(w+2^n) + r' - r.$$

□

定理 2. $0 \leqq x, y < w$, $2^{n-1} < w < 2^n$, $m = 2^n$, $\gcd(m, w) = 1$ を満たす整数 x, y, w, m が与えられたとき, アルゴリズム 3 は MultMon 命令を 2 回呼び出し, MultMonDiv 命令を擬似的に実行する.

証明. 補題 1, 2, 3 より, 証明を導くことができる. □

5.2 ハードウェアによるアプローチ

本節では, MultMon 命令を変更し, MultMonDiv 命令を直接的に計算するアルゴリズム 4 を示す. 実際に MultMon 命令が Montgomery [7] によって提案された標準的なモンゴメリ乗算を実装している場合, MultMon 命令は既にモンゴメリ乗算の商の値を持っているため, 僅かな変更でモンゴメリ乗算の商を計算できる. 変更点は, モンゴメリ乗算の商を保存する Step 2.(c) を挿入し, 剰余とともに商を出力する機能を Step 4 に追加するだけである.

アルゴリズム 4: 変更した MultMon 命令による MultMonDiv 命令

入力: x, y, w, ただし $0 \leqq x, y < w, m = 2^n, \gcd(w, m) = 1, w' = -w^{(-1)} \bmod 2$.
出力: q, r.

1. $q \leftarrow 0, r \leftarrow 0$
2. For i from 0 to $(n-1)$ do the following:
   (a) $u_i \leftarrow (r_0 + x_i y_0)w' \bmod 2$
   (b) $q \leftarrow q + u_i 2^i$
   (c) $r \leftarrow (r + 2r_i + u_i w)/2$
3. If $r \geqq w$ then $r \leftarrow r - w$
4. **Return** (q, r)

6. 実験結果

6.1 検　　証

実験では, IC カード用のエミュレータに提案手法を実装し, 2048 ビットのモンゴメリ乗算による乗算およびべき乗算を検証した. 実験環境上のモンゴメリ乗算器のビット長は 1024 ビットであり, 2048 ビットのモジュラスのきっかり半分のビット長であった. 従って, 定理 2 の仮定より厳しい環境において, 実験により, 提案手法を検証した.

6.2 実装上の問題

$2n$ ビットの整数表現.

実験では, z の値を $(2^n - 1)$ に設定し, 提案手法を実装した. 理由は二つあり, 一つはモンゴメリ乗算に設定するモジュラス n_1 と z をすべて奇数に設定できることであり, もう一つは提案する $2n$ ビットの整数表現へ容易に変換できることである. $z = 2^n - 1$ の場合におけるモンゴメリ乗算の $2n$ ビットのモジュラスの整数表現は, 以下の方程式により, 変換できる.

$$\begin{aligned} N &= n_1' 2^n + n_0' \\ &= n_1(2^n - 1) + n_0 2^n \end{aligned}$$

従って, n_1 と n_0 は以下の方程式から計算できる.

$n_1 := 2^n - n_0'$,

$n_0 := n_1' - n_1 + 1$

モジュラス.

定理 2 が仮定する, 2^{n-1} より大きい値をもつモジュラスが得られない場合がある. 例えば, $w = (2^n - 1)$ の場合には, $n_1 < 2^{n-1}$ となるモジュラス n_1 が得られる. 解決手段としては, アルゴリズム 2 を僅かに変更すればよい. n_1 の値が $0 < n_1 < 2^{n-1}$ であるとき, 定理 2 の仮定を満たすよう, アルゴリズム 2 において, n_1 の代わりに $(m - n_1)$ をモジュラスとして利用する. モジュラスに $(m - n_1)$ を用いても, n_1 をモジュラスとするモンゴメリ乗算の商と剰余が, 次の方程式により計算できる.

$$xy = q(m - w) + rm$$
$$= (-q)w + (q + r)m$$

入力値の補正.

アルゴリズム 2 の入力値が定理 2 の仮定 $(0 < x, y < 2^n)$ を満たさない場合がある. このときは, 入力値とモンゴメリ乗算の剰余を補正し, 解決した. 入力値 x, y に対し, 方程式 $x' := x \pm im$, $y' := y \pm im$, (ただし, i, j は整数) から, 定理 2 の仮定を満たす整数 x', y' が得られる. $xy \cdot m^{-1} \mod w = r$, とすると,

$$(x + im)(y + jm) \cdot m^{-1} = r + jx + iy + ijm \pmod{w}.$$

が成り立つ. 従って, 次の方程式

$$r = (x + im)(y + jm) \cdot m^{-1} - jx - iy - ijm \pmod{w}$$

が得られる.

中間値の補正.

計算中に得るモンゴメリ乗算の商 q と剰余 r の絶対値が減少しない場合がある. このとき, 二つのベクトル $(z, -m)$ と $(n_0 - z, n_1 + m)$ を用い, 次の方程式により, 絶対値の少ない整数 q' と r' を計算する.

$$(q', r') = (q, r) + i(z, -m) + j(n_0 - z, n_1 + m),$$

ただし, i と j は整数である. $z = 2^n - 1$, $m = 2^n$ と設定する場合, 二つのベクトルがとりうる角度は $-m/z \approx -1$, $0 \leqq (n_1 + m)/(n_0 - z) \leqq 3/4$ であるので, ベクトル $(z, -m)$ はもう一方のベクトル $(n_0 - z, n_1 + m)$ と独立である. $|q| < m$ かつ $|r| < m$ となるように補正する, q, r のリダクションの動きを図 2 に示す. この方針では, 次の 4 通りに場合分けをする.

case A: $0 \leqq q, r$ かつ, $m \leqq q \leqq z + m$ または $m \leqq r \leqq z + m$ を満たす場合, (q, r) に $(q - (n_0 - z), r - (n_1 + m))$ を代入し, (q, r) を出力, または, case A または case D へ進む.

case B: $0 \leqq q, r$ かつ, $m \leqq q \leqq z + m$ または $-(z + m) \leqq r \leqq -m$ を満たす場合, (q, r) へ $(q - m, r + z)$ を代入し, (q, r) を出力する.

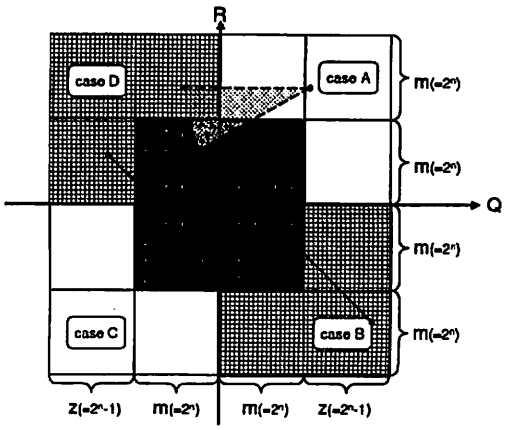

図 2 モンゴメリ乗算の商と剰余の補正

case C: $q, r < 0$ かつ, $-(z + m) \leqq q \leqq -m$ または $-(z + m) \leqq r \leqq -m$ を満たす場合, (q, r) へ $(q + (n_0 - z), r + (n_1 + m))$ を代入し, (q, r) を出力, または case B または case C へ進む.

case D: $0 \leqq q, r$ かつ, $-(z + m) \leqq q \leqq -m$ または $m \leqq r \leqq z + m$ を満たす場合, (q, r) へ $(q + m, r - z)$ を代入し, (q, r) を出力する.

実際には, q, r がとる値域は限定されており, 常に $-3 \cdot 2^n < q < 5 \cdot 2^n$, $-2 \cdot 2^n < r < 2^n$ を満たす. q, r は case A から case D が定義する範囲外の値をとりえるが, 上記の方針における範囲の拡大は容易である.

ビット長の限界.

補題 2 と補題 3 では, モンゴメリ乗算の剰余 r' に対して $r' = \mathrm{MultMon}(x, y, w + 2^n)$ が成り立ち, かつモンゴメリ乗算の商 q' が方程式 $xy = q'(w + 2^n) + r'm$ を満たすことを仮定する. しかし, ハードウェア実装されたモンゴメリ乗算のビット長が n 丁度である場合や, 入力できるモジュラスのビットサイズに制限がある場合には, MultMon 命令は $(w + 2^n)$ をモジュラスとして取り扱えない. このとき, モジュラスに $(w + 2^n)$ を設定する代わりに, $(w \pm 2^{n-2})$ を用いる方法がある. $(w \pm 2^{n-2})$ の場合におけるアルゴリズムは付録 2. に示す.

7. ま と め

本稿では, n ビットのモンゴメリ乗算を実装するコプロセッサを利用し, $2n$ ビットのモンゴメリ乗算を実現する計算方式を示した. 従来技術である Fischer らのアルゴリズムは, 古典的な剰余演算に基づいて設計されたため, モンゴメリ乗算器に適用できなかった. そこで, 本稿では, モンゴメリ乗算器が利用できるよう, まず Fischer らが提案したアルゴリズムを修正し, 新たにモンゴメリ乗算の商を定義した. 修正したアルゴリズムはモンゴメリ乗算の商を必要とする. 次に, モンゴメリ乗算器用に,

入力された $2n$ ビットの整数に対する新しい整数表現を提案した．新しい整数表現は，モンゴメリ乗算の条件を満たすようにモジュラスを常に奇数にしつつ，$2n$ ビットの整数を効率的に変換かつ逆変換する．最後に，提案するアルゴリズムが必要とするモンゴメリ乗算の商を効率的に計算するアルゴリズムを示した．

モンゴメリ乗算は，IC カード上のコプロセッサなど，公開鍵暗号系において広く実利用されている．本稿では，モンゴメリ乗算を実装するコプロセッサを用い，新たに定義したモンゴメリ乗算の商の擬似的な計算手法を提案した．また，モンゴメリ乗算器が変更可能である場合は，モンゴメリ乗算の商を直接的に計算するアルゴリズムを提案した．擬似的な計算手法では，モンゴメリ乗算器を 2 回呼出す必要があり，直接的に計算する場合には，変更したモンゴメリ乗算器の呼出しは 1 回でよい．最後に，実験により，提案手法が $2n$ ビットのモンゴメリ乗算を計算することを確認した．

表 1　モンゴメリ乗算の商を擬似的に計算する MultMon 命令

	命令数 (平均)	
モジュラス	$0 < w < 2^{n-1}$	$2^{n-1} < w < 2^n$
加算/減算	5.5	7.5
MultMon 命令	2	2
MultMonDiv命令	0	0

表 2　モンゴメリ乗算の商を直接的に計算する MultMon 命令

	命令数 (平均)	
モジュラス	$0 < w < 2^{n-1}$	$2^{n-1} < w < 2^n$
加算/減算	0	2
MultMon 命令	0	0
MultMonDiv命令	1	1

今後の研究課題は，モンゴメリ乗算器の呼び出し回数の最適化である．例えば，古典的な剰余演算の場合，Chevallier-Mames らは，Fischer らの提案手法をより最適化した手法を示した．本稿では，モンゴメリ乗算の条件から，Fischer らの提案手法を拡張したが，今後は Chevallier-Mames らの手法のモンゴメリ乗算への拡張を検討する．

文　献

[1] J.-C. Bajard, L.-S. Didier, P. Kornerup: "An RNS Montgomery Modular Multiplication Algorithm", IEEE Computer Society, pp. 234–239 (1997).

[2] B. Chevallier-Mames, M. Joye, P. Paillier: "Faster Double-Size Modular Multiplication From Euclidean Multipliers", CHES2003, vol. 2779 of Lecture Notes in Computer Science, Springer-Verlag, pp. 214–227 (2003).

[3] W. Fischer, J.-P. Seifert: "Increasing the bitlength of crypto-coprocessors", CHES2002, vol. 2523 of Lecture Notes in Computer Science, Springer-Verlag, pp. 71–81 (2003).

[4] S. Kawamura, M. Koike, F. Sano, A. Shimbo: "Cox-Rower Architecture for Fast Parallel Montgomery Multiplication", EUROCRYPT2000, vol. 1807 of Lecture Notes in Computer Science, Springer-Verlag, pp. 523–538 (2000).

[5] C.K. Koc: "Montgomery reduction with even modulus", IEE Proceedings: Computer and Digital Techniques, vol. 141, no.5, pp. 314–316 (1994).

[6] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone: Handbook of Applied Cryptography, CRC Press (1996).

[7] P.L. Montgomery: "Modular multiplication without trial division", Mathematics of Computation, vol. 44, no. 170, pp. 519–521 (1985).

[8] D. Naccache, D. M'Raïhi: "Arithmetic co-processors for public-key cryptography : The state of the art", CARDIS, pp. 18–20, (1996).

[9] P. Paillier: "Low-Cost Double-Size Modular Exponentiation or How to Stretch Your Cryptoprocessor", PKC, vol. 1560 of Lecture Notes in Computer Science, Springer-Verlag, pp. 223–234 (1999).

[10] K.C. Posch, R. Posch: "Modulo Reduction in Residue Number Systems", IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 5, pp. 449–454 (1995).

[11] J.-J. Quisquater, C. Couvreur: "Fast decipherment algorithm for RSA public-key cryptosystem", Electronics Letters, vol. 18, no. 21, pp. 905–907 (1982).

[12] J.-J. Quisquater, B. Schneier: "Smart Card Crypto-Coprocessors for Public-Key Cryptography", CARDIS, vol. 1820 of Lecture Notes in Computer Science, Springer-Verlag, pp. 386–394 (2000).

[13] RSA chalenges: http://www.rsasecurity.com/rsalabs/

[14] R.L. Rivest, A. Shamir, L. Adleman: "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM, vol. 21, no. 2, pp. 120–126 (1978).

付　　録

1. 修正した Fischer らのアルゴリズム

本節では, 定理 1 の証明, 即ち, アルゴリズム 2 が, $ABM^{(-1)} \pmod N$(ただし, $0 \leqq A, B < N$, $2^{2n-1} < N < 2^{2n}$, $M = 2^{2n}$) を出力することを証明する.

証明. まず, $2n$ ビットからなる整数 A, B, N を次の方程式で表す.

$$A = a_1 z + a_0 m,\ B = b_1 z + b_0 m,\ N = n_1 z + n_0 m$$

ただし, z は奇数かつ $2^{n-1} \leqq z < 2^n$ であり, $m = 2^n$ とする. このとき, 以下の式が成り立つ.

$$\begin{aligned}
AB &= (a_1 z + a_0 m)(b_1 z + b_0 m) \\
&= a_1 b_1 zz + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1 (q_1 n_1 + r_1 m) z + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1 r_1 zm - a_1 q_1 n_0 m + a_1 b_0 zm + a_0 b_1 zm + a_0 b_0 mm \\
&= a_1 r_1 zm - a_1 (q_2 z + r_2 m) m + a_1 b_0 zm + a_0 b_1 zm \\
&\quad + a_0 b_0 mm \\
&= a_1 (r_1 - q_2 + b_1) zm - a_1 r_2 mm + a_0 b_1 zm + a_0 b_0 mm \\
&= (q_3 n_1 + r_3 m) zm - a_1 r_2 mm + a_0 b_1 zm + a_0 b_0 mm \\
&= (q_3 n_1 + r_3 m) zm - a_1 r_2 mm + (q_4 n_1 z + r_4 m) zm \\
&\quad + a_0 b_0 mm \\
&= (r_3 + r_4) zmm - a_1 r_2 mm - (q_3 + q_4) mm + a_0 b_0 mm \\
&= (r_3 + r_4) zmm - a_1 r_2 mm - (q_5 z + r_5 m) mm + a_0 b_0 mm \\
&= (r_3 + r_4) zmm - (q_6 z + r_6 m) mm - (q_5 z + r_5 m) mm \\
&\quad + a_0 b_0 mm \\
&= (r_3 + r_4) zmm - (q_6 z + r_6 m) mm - (q_5 z + r_5 m) mm \\
&\quad + (q_7 z + r_7 m) mm \\
&= (r_3 + r_4 - q_5 - q_6 + q_7) zmm - (r_5 + r_6 - r_7) mmm \\
&= \{(r_3 + r_4 - q_5 - q_6 + q_7) z - (r_5 + r_6 - r_7) m\} m^2
\end{aligned}$$

従って, 次の方程式

$$ABM^{-1} = (r_3 + r_4 - q_5 - q_6 + q_7) z - (r_5 + r_6 - r_7) m$$

が成り立ち, アルゴリズム 2 は, たしかに $2n$ ビットのモンゴメリ乗算を計算する. □

2. ビット長が限定されたモンゴメリ乗算器を用いた場合のソフトウェアによるアプローチ

アルゴリズム 3 ではモジュラスに $(w + 2^n)$ を用いたが, 代わりに $(w \pm 2^{n-2})$ を用いることができる. モンゴメリ乗算器のビット長が丁度 n の場合などに, このモンゴメリ乗算の商を計算するアルゴリズムは有効である. ここでは, モジュラスのビット長が n を越えないよう, モジュラス w の値に応じて場合分けを行い, $(2^{n-1} < w < 2^{n-1} + 2^{n-2})$ の場合はアルゴリズム 5, $(2^{n-1} + 2^{n-2} < w < 2^n)$ の場合はアルゴリズム 6 を実行する.

アルゴリズム 5: MultMonDiv 命令の実装 (1)

入力: x, y, w ただし $0 \leqq x, y < w$, $2^{n-1} + 2^{n-2} < w < 2^n$, $m = 2^n$, $\gcd(w, m) = 1$.
出力: q, r.

1. $r \leftarrow$MultMon(x, y, w)
2. $r' \leftarrow$MultMon$(x, y, w - 2^{n-2})$
3. $tmp \leftarrow xy + rw - r'(w - 2^{n-2}) \pmod{2^4}$
4. if tmp $\geqq 2^3$ then $tmp \leftarrow tmp - 2^4$.
5. $q \leftarrow tmp \times (w - 2^{n-2}) + 4(r' - r)$
6. **Return** (q, r)

アルゴリズム 6: MultMonDiv 命令の実装 (2)

入力: x, y, w ただし $0 \leqq x, y < w$, $2^{n-1} < w < 2^{n-1} + 2^{n-2}$, $m = 2^n$, $\gcd(w, m) = 1$.
出力: q, r.

1. $r \leftarrow$MultMon(x, y, w)
2. $r' \leftarrow$MultMon$(x, y, w + 2^{n-2})$
3. $tmp \leftarrow xy - rw + r'(w + 2^{n-2}) \pmod{2^4}$
4. if tmp $\geqq 2^3$ then $tmp \leftarrow tmp - 2^4$.
5. $q \leftarrow tmp \times (w + 2^{n-2}) + 4(r - r')$
6. **Return** (q, r)