(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第4662802号
(P4662802)
(45)発行日　平成23年3月30日(2011.3.30)　(24)登録日　平成23年1月14日(2011.1.14)

(51)Int.Cl.　　F I
G09C 1/00 (2006.01)　　G09C 1/00 620A
G06F 7/72 (2006.01)　　G06F 7/72

請求項の数 9　(全 37 頁)

(21)出願番号　特願2005-99980(P2005-99980)
(22)出願日　平成17年3月30日(2005.3.30)
(65)公開番号　特開2006-276786(P2006-276786A)
(43)公開日　平成18年10月12日(2006.10.12)
審査請求日　平成19年12月19日(2007.12.19)

(73)特許権者　000005223
富士通株式会社
神奈川県川崎市中原区上小田中４丁目１番１号
(74)代理人　100078868
弁理士　河野　登夫
(72)発明者　伊藤　孝一
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内
(72)発明者　向田　健二
神奈川県川崎市中原区上小田中４丁目１番１号　富士通株式会社内

審査官　青木　重徳

最終頁に続く

(54)【発明の名称】計算方法、計算装置及びコンピュータプログラム

(57)【特許請求の範囲】
【請求項１】
モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを備えた計算装置を用いて計算する計算方法において、
前記計算装置は、
2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納するステップと、
レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納するステップと、
レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算するステップと
を実行することを特徴とする計算方法。
【請求項２】
計算した同値を用いて、べき乗剰余処理を実行することを特徴とする請求項１に記載の計算方法。
【請求項３】
モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算する計算装置において、
レジスタと、

剰余の法ｎの負数を前記レジスタに格納する手段と、
レジスタに格納されている値を桁上がり方向へ１ビットシフトする処理を、レジスタからあふれる最上位ビットが０になるまで繰り返す手段と、
レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算する手段と
を備えることを特徴とする計算装置。
【請求項４】
モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算する計算装置において、
１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタと、
値Ａ及びＢ、並びに有効ワード長がｍである剰余の法ｎに対し、$2^{-m*k}\times A\times B$（ｍｏｄ　ｎ）として定義されるモンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ を実行する演算手段と、
剰余の法ｎの負数をレジスタに格納する手段と、
レジスタに格納されている値を桁上がり方向へ１ビットシフトするシフト処理を、レジスタからあふれる最上位ビットが０になるまで繰り返す手段と、
前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ＲＥＧ）ｎ を実行して、その結果をレジスタに格納する処理を、$2^{p-1}<m\times k\leqq 2^{p}$ を満たす整数であるｐ回繰り返す手段と、
$2^{p}>m\times k$である場合に、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ｇ）ｎ を実行して、その結果をレジスタに格納する手段と（ただし$g=2^{k*G(p,m,k)}$、かつＧ（ｐ，ｍ，ｋ）＝$2\times m-2^{p}/k$）、
レジスタに格納されている値を、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値として出力する手段と
を備えることを特徴とする計算装置。
【請求項５】
複数のレジスタと、
ｍ個のワードを有する第１のレジスタ及びｍ個以上のワードを有する第２のレジスタに、夫々ｎ及び０を格納する手段と、
第２のレジスタに格納されている値から、第１のレジスタに格納されている値を減じて、剰余の法ｎの負数を計算する手段と
を更に備えることを特徴とする請求項４に記載の計算装置。
【請求項６】
レジスタに剰余の法ｎを格納する手段と、
レジスタに格納されている値の補数を、剰余の法ｎの負数として計算する手段と
を更に備えることを特徴とする請求項４に記載の計算装置。
【請求項７】
レジスタに剰余の法ｎを格納する手段と、
レジスタに格納されている値を反転させる手段と、
レジスタに格納されている値の最下位ビットを１として、剰余の法ｎの負数を計算する手段と
を更に備えることを特徴とする請求項４に記載の計算装置。
【請求項８】
前記シフト処理は、レジスタに格納されている値に該値を加算する加算処理であり、
前記シフト処理によりレジスタからあふれる最上位ビットは、前記加算処理により発生したキャリー値として検出する
ことを特徴とする請求項４乃至請求項７のいずれかに記載の計算装置。
【請求項９】
１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを備える

コンピュータに、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算させるコンピュータプログラムにおいて、
コンピュータに、2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納させる手順と、
コンピュータに、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納させる手順と、
コンピュータに、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算させる手順と
を実行させることを特徴とするコンピュータプログラム。

【発明の詳細な説明】

【技術分野】

【０００１】
本発明は、モンゴメリ乗算剰余演算にて用いられるモンゴメリ変換パラメータに関する値を計算する計算方法、該計算方法を適用した計算装置、及び該計算装置を実現するためのコンピュータプログラムに関し、特に計算速度を高速化する計算方法、計算装置及びコンピュータプログラムに関する。

【背景技術】

【０００２】
今後の情報化社会の発展に伴い、電子マネー、住民基本台帳ネットワーク等の情報ネットワークを利用したサービスが普及すると予想される。これらのサービスを安全に運用するためには、情報セキュリティ技術が必須であり、情報セキュリティの基盤技術として暗号技術が用いられる。暗号技術を用いることで、暗号、デジタル署名、認証等の機能を実現し、個人情報を第三者からの不正なアクセスから防御することができる。

【０００３】
暗号技術を実現するための暗号方式は現在まで様々な方式が知られており、これらは共通鍵暗号方式と公開鍵暗号方式との２種類に大別される。共通鍵暗号方式と呼ばれるものは、暗号化と復号とで同一の鍵(共通鍵)を用いる方式であり、この共通鍵を送信者及び受信者以外の第三者にわからない情報とすることで安全性を保つ方式である。公開鍵暗号方式とは、暗号化と復号とで異なる鍵を用いる方式であり、暗号化を行うための鍵(公開鍵)を一般に公開する代わりに、暗号文を復号するための鍵(秘密鍵)を受信者のみの秘密情報とすることで安全性を保つ方式である。共通鍵暗号方式を用いる場合、前述の共通鍵を送受信者以外の第三者にわからない安全な形で共有する必要がある。これに対し公開鍵暗号方式は、送受信者間で秘密情報を共有する必要がないというメリットを有するが、処理を行うための計算量が共通鍵暗号方式と比べて非常に大きいというデメリットを有する。よって、公開鍵暗号方式においては、計算処理の高速化が大きな課題となる。

【０００４】
公開鍵暗号方式は、ＲＳＡ暗号、楕円曲線暗号が代表的な方式として知られている。ＲＳＡ暗号においてはべき乗剰余演算を用いた処理が、楕円曲線暗号においては点のスカラー倍算と呼ばれる演算を用いた処理が夫々行われる。これら２つの演算のいずれについても、剰余の法を示す整数ｎと、$0 \leqq a, b < n$となる整数ａ，ｂとを用いた式$y = a \times b \pmod{n}$にて示す乗算剰余演算が基本演算として用いられる。

【０００５】
ところが乗算剰余演算をそのままハードウェア又はソフトウェアにて実装した場合、処理時間が大きく処理効率が悪くなる。そのため乗算剰余演算の代わりに下記の式にて示す整数ａ，ｂ，ｎを用いたモンゴメリ乗算剰余と呼ばれる演算法を用いて計算するのが一般的に行われており、下記の式に示すモンゴメリ乗算剰余演算を用いることで、通常の乗算剰余演算より高速な処理を実現することが可能である。なお下記の式及び以降の説明にお

いて、「＊」は、乗算記号「×」を示すものとする。

【０００６】

$y = a \times b \times R^{-1} \pmod{n}$

ただし、ｎ：剰余の法を示す整数

ａ，ｂ：$0 \leqq a, b < n$となる整数

Ｒ：2^{m*k} にて示される定数

ｋ：１ワードあたりのビット長

ｍ：ｎを表現するために必要なワードの最小個数

【０００７】

図１３は、モンゴメリ乗算剰余演算のアルゴリズムを示す説明図である。なお図１３に示すアルゴリズムにおいて、$x = (x_{m-1}, \cdots, x_1, x_0)$は、整数値ｘを、ｍ個のワード値$x_i$（$i = m-1, \cdots, 1, 0$，$0 \leqq x_i < 2^k$）として表現する形式を示す。図１３に示す様に夫々ｍワード値にて示されるａ，ｂ，ｎに基づいて、ｍワード値で示される値ｙを算出する場合のモンゴメリ乗算剰余演算$y = a \times b \times R^{-1} \pmod{n}$を、以降の説明では、ｙ＝ＲＥＤＣ（ａ，ｂ）ｎ 又は単にＲＥＤＣと表記する。また図１３を含む以降の図面及び以降の説明において、「：＝」とは、右辺の数値又は数式を左辺に代入することを示す。

【０００８】

上述した様にモンゴメリ乗算剰余演算は、$a \times b \times R^{-1} \pmod{n}$であり、通常の乗算剰余演算$a \times b \pmod{n}$とは異なる演算を行う。よってべき乗剰余演算を正しく実行するためには、モンゴメリ乗算剰余に対して与える入力データをモンゴメリ系と呼ばれるデータに変換する必要がある。通常の乗算剰余演算に与える任意の入力データをｘ、ｘをモンゴメリ系に変換したデータをｘ’とし、ｘからｘ’への変換(モンゴメリ変換)をｘ’＝Ｍｏｎｔ（ｘ）として表し、ｘ’からｘへの変換(モンゴメリ逆変換)をｘ＝Ｍｏｎｔ$^{-1}$（ｘ’）と表した場合、これらは下記の式にて与えられる。

【０００９】

モンゴメリ変換：$x' = Mont(x) = x \times R \pmod{n}$

モンゴメリ逆変換：$x = Mont^{-1}(x') = x' \times R^{-1} \pmod{n}$

【００１０】

上記式にて示したモンゴメリ変換及びモンゴメリ逆変換は、ＲＥＤＣを用いた下記の式にて示すことができる。ただしＨはＨ＝Ｒ^2 （ｍｏｄ　ｎ）で表されるモンゴメリ変換パラメータと呼ばれる値であり、事前計算により求められる。

【００１１】

モンゴメリ変換：$x' = REDC(x, H)_n = x \times R^2 \times R^{-1} = x \times R \pmod{n}$

ただし、$H = R^2 \pmod{n}$

モンゴメリ逆変換：$x = REDC(x', 1)_n = x' \times 1 \times R^{-1} = x' \times R^{-1} \pmod{n}$

【００１２】

上述した数式に基づくモンゴメリ乗算剰余を用いたべき乗剰余演算のアルゴリズムについて説明する。図１４は、モンゴメリ乗算剰余演算を用いたべき乗剰余演算のアルゴリズムを示す説明図である。図１４は、バイナリ法と呼ばれるべき乗剰余演算に基づくモンゴメリ乗算剰余演算のアルゴリズムを示しており、入力値ａ，ｄ，ｎからべき乗剰余演算結果$y = a^d \pmod{n}$を計算する。図１４における１行目の処理は、ｙの初期値として１を与えることを示している。２行目の処理は、モンゴメリ変換パラメータ$H = R^2 \pmod{n}$を計算することを示している。３行目の処理は、ｙとａとに対しモンゴメリ変換を行いｙ’とａ’とを得ることを示している。４～７行目のループは、ｄのビット値に応じてモンゴメリ乗算剰余を１回又は２回繰り返す処理を、ｄの最下位ビットから最上位ビットまで繰り返すことを示している。８行目の処理は、４～７行目のループで計算されたｙ’に対し、モンゴメリ逆変換を行うことで、最終的な演算結果ｙを得ることを示し

ている。

【００１３】

図１４に示したアルゴリズムにおいて、２行目で行われるモンゴメリ変換パラメータH＝R^2（ｍｏｄ　ｎ）の計算方法について説明する。図１５は、モンゴメリ変換パラメータの計算方法のアルゴリズムを示す説明図である。図１５に示すモンゴメリ変換パラメータの計算方法は、加算、比較及び減算を繰り返すことにより、R＝2^x とする場合のH＝R^2（ｍｏｄ　ｎ）を計算する方法である。１行目の処理は、H＝R（ｍｏｄ　ｎ）を計算することを示している。H＝R（ｍｏｄ　ｎ）の算出法は様々な方法があるが、例えばR＝2^xに対してnの有効ビット長がｘである場合、R（ｍｏｄ　ｎ）＝０－ｎにより簡単に計算できる。２～５行目のループは、H＝R（ｍｏｄ　ｎ）に対し、H＋Hを計算した後、結果がｎ以上である場合にｎを減算することで、H＋H（ｍｏｄ　ｎ）の加算剰余（２倍剰余）を行っている。なおH＋Hの計算は、左１ビットシフト演算でも実現することが可能である。図１５に示すアルゴリズムでは、上述した加算剰余演算をｘ回繰り返すことにより、R×2^x（ｍｏｄ　ｎ）＝R^2（ｍｏｄ　ｎ）を算出する。

【００１４】

しかしながら図１５に示したモンゴメリ変換パラメータの計算方法のアルゴリズムは、２～５行目で加算剰余をｘ回繰り返すため、処理速度が遅いという欠点を有する。例えばｎが１０２４ｂｉｔであるＲＳＡ演算の場合では、R＝2^{1024}であるため、１０２４回の加算剰余演算を行う必要があるため、計算量が膨大となり処理速度が遅くなる。

【００１５】

そこでＲＥＤＣ演算、シフト算及び減算を組み合わせることにより、モンゴメリ変換パラメータH＝R^2（ｍｏｄ　ｎ）の計算速度を高速化する幾つかの方法が提案されている。これらの方法を以下に従来法１～３として説明する。なお以下の従来法１乃至従来法３の説明において、１ワードあたりのビット長をｋとし、ｍワード値で示された値をｎとし、ｎの最上位から連続する「０」の個数をｑとして示す。例えばｋ＝８の場合、ｎのビット列が「００１０１０１１　１１００１１１１」であれば、ｍ＝２，ｑ＝２であり、ｎのビット列が「１０００１００１　１１１００１１０　１１１００１０１」であれば、ｍ＝３，ｑ＝０である。

【００１６】

従来法１.

図１６は、従来法１におけるモンゴメリ変換パラメータの計算方法を示すフローチャートである。図１６に示す従来法１では、剰余の法ｎを入力し、R^2（ｍｏｄ　ｎ）を出力するものとする。ただし、R＝2^{m*k}（ｍｏｄ　ｎ）である。従来法１は、主にステップＡ１及びステップＢ１にて構成される。ステップＡ１は、シフト算及び減算を用いて、H_0＝2^v×R（ｍｏｄ　ｎ）を計算するステップである。ただし、ｖは自然数である。ステップＢ１は、ＲＥＤＣ演算を用いてH_0からH＝R^2（ｍｏｄ　ｎ）を計算するステップである。

【００１７】

ステップＡ１のステップＳ１０１では、第１レジスタＲＥＧ１及び第２レジスタＲＥＧ２に対し、初期値として夫々「ｎ」及び「０」を与える。なおｎの有効ワード長はｍであり、第１レジスタＲＥＧ１に右詰で格納した初期値ｎの最上位ビットから連続する「０」の個数をｑとして示す。なお以降の説明において第１レジスタＲＥＧ１に格納した値をＲＥＧ１として示し、第２レジスタＲＥＧ２に格納した値をＲＥＧ２として示す。

【００１８】

ステップＡ１のステップＳ１０２では、第１レジスタＲＥＧ１に対し、左１ビットシフト処理をｑ回繰り返し、ＲＥＧ１＝ｎ’＝ｎ×2^q を計算する。

【００１９】

ステップＡ１のステップＳ１０３では、ＲＥＧ２－ＲＥＧ１として算出した値を第２レジスタＲＥＧ２に格納することで、ＲＥＧ２＝2^{m*k}（ｍｏｄ　ｎ’）とする。

【００２０】

ステップＡ１のステップＳ１０４では、第２レジスタＲＥＧ２に対する左１ビットシフト処理と、ＲＥＧ２≧ＲＥＧ１の真偽判定と、ＲＥＧ２≧ＲＥＧ１が真である場合にＲＥＧ２－ＲＥＧ１の演算結果を第２レジスタＲＥＧ２に格納する処理とを、ｖ＋ｑ回繰り返し、ＲＥＧ２＝$2^{m*k+v+q}$とする。ただし、ｖは、ｖ≧１であり、かつｍ，ｋに対し（ｍ×ｋ）／ｖが２のべき乗となる整数である。

【００２１】

ステップＡ１のステップＳ１０５では、第１レジスタＲＥＧ１及び第２レジスタＲＥＧ２に対し、右１ビットシフト処理をｑ回繰り返し、ＲＥＧ１＝ｎ，ＲＥＧ２＝$H_0 = 2^{m*k+v}$（ｍｏｄ　ｎ）を計算する。

【００２２】

ステップＢ１のステップＳ１０６では、ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）ｎとして示されるＲＥＤＣ演算の結果を第２レジスタＲＥＧ２に格納する処理をｐ回繰り返すことにより、ＲＥＧ２＝Ｈ＝2^{2*m*k}（ｍｏｄ　ｎ）＝R^2（ｍｏｄ　ｎ）を計算する。ただし、ｐは、ｐ＝$\log_2$（（ｍ×ｋ）／ｖ）を満たす整数であり、ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）ｎは、モンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ＝2^{-m*k}×Ａ×Ｂ（ｍｏｄ　ｎ）を示す。

【００２３】

そしてステップＳ１０７では、計算した結果であるＲＥＧ２＝R^2（ｍｏｄ　ｎ）を出力し、処理を終了する。

【００２４】

図１７は、従来法１におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図１７では、図１６を用いて示した従来法１の計算方法に必要な演算回数を演算の種類及びステップ別に示している。なお図１７において、ＳＦＴは１ビットシフトを行うシフト演算を、ＳＵＢは減算を、ＣＭＰは比較演算を、ＲＥＤＣはモンゴメリ乗算剰余演算をそれぞれ示す。

【００２５】

ステップＳ１０６において、ｐは、ｐ＝$\log_2$（（ｍ×ｋ）／ｖ）を満たす整数であるとの条件を示したが、これを満たすためには、（ｍ×ｋ）／ｖが整数ｘを用いて（ｍ×ｋ）／ｖ＝2^xと示せる値、即ち２のべき乗となる値でなければならないという制約がある。この制約により、従来法１では、ｖの値の選択が制限されるため、ｎの有効ビット長によっては、ｖの値を大きくする必要がある。図１７に示す図表から、ＳＦＴ、ＳＵＢ及びＣＭＰの計算回数は、ｖに依存するため、ｖを大きくすることにより、全体の計算量が大きくなる。

【００２６】

次に図１７に示した図表に基づいて、従来法１における計算方法の演算回数の例を示す。

例１－１．１０２４ビットのＲＳＡ暗号の計算に適用

上記条件より、ｎは１０２４ビットである。１ワード＝３２ビットとするとｋ＝３２であり、ｎの有効ワード長ｍ＝３２となる。１ワードあたりのビット長ｋとｎの有効ワード長ｍとを乗算したｋ×ｍとｎの全ビット数とが一致することから、ｎの最上位ビット＝１となりｑ＝０となる。またｍ×ｋ＝１０２４であるので、ｖ＝１，２，４，…，１０２４を選択することが可能である。ｖ＝１の場合、ＳＦＴが０×５＋１＝１回、ＳＵＢが０．５×（０＋１）＋１＝１．５回、ＣＭＰが０＋１＝１回、そしてＲＥＤＣがｐ＝$\log_2$（（３２×３２）／１）＝１０回となる。

【００２７】

例１－２．１６３ビットの楕円曲線暗号の計算に適用

上記条件より、ｎは１６３ビットである。１ワード＝８ビットとするとｋ＝８であり、ｎの有効ワード長ｍ＝２１となる。ｎをビット長＝８、有効ワード長ｍ＝２１とすると、最上位に位置するｍ×ｋ－１６３＝２１×８－１６３＝５ビットが０となり、ｑ＝５となる。またｍ×ｋ＝１６８であるので、ｖ＝２１，４２，８４，１６８を選択することが可

能である。ｖ＝２１の場合、ＳＦＴが４×５＋２１＝４１回、ＳＵＢが０．５×（５＋２１）＋１＝１４回、ＣＭＰが５＋２１＝２６回、そしてＲＥＤＣがｐ＝ｌｏｇ$_2$（（２１×８）／２１）＝３回となる。

【００２８】

このような従来法１に示す計算方法は、例えば特許文献１、特許文献２及び特許文献３に示されている。

【００２９】

従来法２．

図１８は、従来法２におけるモンゴメリ変換パラメータの計算方法を示すフローチャートである。図１８に示す従来法２では、剰余の法ｎを入力し、R^2（ｍｏｄ　ｎ）を出力するものとする。ただし、$R=2^{m*k}$（ｍｏｄ　ｎ）である。従来法２は、主にステップＡ２及びステップＢ２にて構成される。ステップＡ２は、例えば従来法１に示した処理と同様の処理により、シフト算及び減算を用いて$H_0=2^v \times R$（ｍｏｄ　ｎ）を計算するステップである。ただし、ｖは自然数である。ステップＢ２は、ＲＥＤＣ演算を用いてH_0から$H=R^2$（ｍｏｄ　ｎ）を計算するステップである。

【００３０】

ステップＡ２のステップＳ２０１では、第１レジスタＲＥＧ１及び第２レジスタＲＥＧ２に対し、初期値として夫々「ｎ」及び「０」を与える。なおｎの有効ワード長はｍであり、第１レジスタＲＥＧ１に右詰で格納した初期値ｎの最上位ビットから連続する「０」の個数をｑとして示す。

【００３１】

ステップＡ２のステップＳ２０２では、第１レジスタＲＥＧ１に対し、左１ビットシフト処理をｑ回繰り返し、$REG1=n'=n\times 2^q$を計算する。

【００３２】

ステップＡ２のステップＳ２０３では、ＲＥＧ２－ＲＥＧ１として算出した値を第２レジスタＲＥＧ２に格納することで、$REG2=n'=n\times 2^q$とする。

【００３３】

ステップＡ２のステップＳ２０４では、第２レジスタＲＥＧ２に対する左１ビットシフト処理と、ＲＥＧ２≧ＲＥＧ１の真偽判定と、ＲＥＧ２≧ＲＥＧ１が真である場合にＲＥＧ２－ＲＥＧ１の演算結果を第２レジスタＲＥＧ２に格納する処理とからなる２倍剰余演算を、ｖ＋ｑ回繰り返し、$REG2=2^{m*k+v+q}$とする。ただし、ｖは、ｖ≧１であり、かつｍ，ｋに対し（ｍ×ｋ）／ｖが自然数となる整数である。

【００３４】

ステップＡ２のステップＳ２０５では、第１レジスタＲＥＧ１及び第２レジスタＲＥＧ２に対し、右１ビットシフト処理をｑ回繰り返し、ＲＥＧ１＝ｎ，$REG2=H_0=2^{m*k+v}$（ｍｏｄ　ｎ）を計算する。そして第２レジスタＲＥＧ２に格納されている値を、補助レジスタＲＥＧ０に格納する。

【００３５】

ステップＢ２のステップＳ２０６では、ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）ｎとして示されるＲＥＤＣ演算の結果を第２レジスタＲＥＧ２に格納し、更に（ｍ×ｋ）／ｖのｉ番目のビット値＝１の場合に、ＲＥＤＣ（ＲＥＧ２，ＲＥＧ０）ｎとして示されるＲＥＤＣ演算の結果を第２レジスタＲＥＧ２に格納する処理を、ｉ＝ｐ’－２，…，１，０としてｐ’－１回繰り返すことにより、$REG2=H=2^{2*m*k}$（ｍｏｄ　ｎ）$=R^2$（ｍｏｄ　ｎ）を計算する。ただし、ｐ’は、（ｍ×ｋ）／ｖのビット長を示す整数であり、ＲＥＤＣ（Ａ，Ｂ）ｎは、モンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）$n=2^{-m*k}\times A\times B$（ｍｏｄ　ｎ）を示す。

【００３６】

そしてステップＳ２０７では、計算した結果である$REG2=R^2$（ｍｏｄ　ｎ）を出力し、処理を終了する。

【００３７】

図１９は、従来法２におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図１９では、図１８を用いて示した従来法２の計算方法に必要な演算回数を演算の種類及びステップ別に示している。なお図１９において、ＳＦＴは１ビットシフトを行うシフト演算を、ＳＵＢは減算を、ＣＭＰは比較演算を、ＲＥＤＣはモンゴメリ乗算剰余演算をそれぞれ示す。またW（ｘ）は、ｘの最上位ビットを除く１の個数を示し、ステップＳ２０６において（ｍ×ｋ）／ｖのビット値が１である場合のＲＥＤＣ演算の回数である。例えば、W（（１００００）$_2$）＝０であり、W（（１０００１０１）$_2$）＝２である。ただし、（・・・）$_2$は、２進数を示す記号であり、例えば（１１０１）$_2$＝１３であり、（１１１００）$_2$＝２８である。

【００３８】

ステップＳ２０６にて示した様に、ｐ’は、（ｍ×ｋ）／ｖとして示される整数であることから、従来法２では、従来法１より広い条件でｖの値を設定することができるので、最適なｖの値を設定することで従来法１より少ない計算量でモンゴメリ変換パラメータHを計算することができる。

【００３９】

次に図１９に示した図表に基づいて、従来法２における計算回数の例を説明する。

例２－１．１０２４ビットのＲＳＡ暗号の計算に適用

上記条件より、ｎは１０２４ビットである。１ワード＝３２ビットとするとｋ＝３２であり、ｎの有効ワード長ｍ＝３２となる。１ワードあたりのビット長ｋとｎの有効ワード長ｍとを乗算したｋ＋ｍとｎの全ビット数とが一致することから、ｎの最上位ビット＝１となりｑ＝０となる。またｍ×ｋ＝１０２４であるので、ｖは１０２４の任意の因数（ｆａｃｔｏｒ）から選択することが可能である。Ｖ＝１の場合、ＳＦＴが１回、ＳＵＢが０．５×（１）＋１＝１．５回、ＣＭＰが１回、そしてＲＥＤＣがｐ＝ｌｏｇ$_2$（（３２×３２）／１）＝１０回となる。

【００４０】

例２－２．１６３ビットの楕円曲線暗号の計算に適用

上記条件より、ｎは１６３ビットである。１ワード＝８ビットとするとｋ＝８であり、ｎの有効ワード長ｍ＝２１となる。ｎをビット長＝８、有効ワード長ｍ＝２１とすると、最上位に位置するｍ×ｋ－１６３＝２１×８－１６３＝５ビットが０となり、ｑ＝５となる。またｍ×ｋ＝１６８であるので、ｖは１６８の任意の因数（ｆａｃｔｏｒ）から選択することが可能である。ｖ＝２１の場合、ＳＦＴが４×５＋２１＝４１回、ＳＵＢが０．５×（５＋２１）＋１＝１４回、ＣＭＰが５＋２１＝２６回、そしてＲＥＤＣが（ｍ×ｋ）／ｖ＝（１０００）$_2$からｐ’－１＋W（（ｍ×ｋ）／ｖ）＝４－１＋０＝３回となる。

【００４１】

このような従来法２に示す計算方法は、例えば特許文献４に示されている。

【００４２】

従来法３．

図２０は、従来法３におけるモンゴメリ変換パラメータの計算方法を示すフローチャートである。図２０に示す従来法３では、剰余の法ｎを入力し、R^2（ｍｏｄ　ｎ）を出力するものとする。ただし、$R=2^{m*k}$（ｍｏｄ　ｎ）である。従来法３は、主にステップＡ３、ステップＢ３及びステップＣ３にて構成される。ステップＡ３は、シフト算及び減算を用いて$H_0=2^{m*k+v}$を満たすH_0を計算するステップである。ただし、ｖは自然数であり、かつ（ｍ×ｋ）／ｖが自然数であることを満たす。ステップＢ３は、ＲＥＤＣ演算を用いてH_0から$H=2^{E(p'',m,k)}$（ｍｏｄ　ｎ）を計算するステップである。ただし、ｐ”は、$2^{p''-1}<(m\times k)/v\leqq 2^{p''}$を満たす整数であり、Ｅ（ｐ”，ｍ，ｋ）＝ｍ×ｋ＋ｖ×$2^{p''}$である。ステップＣ３は、２ｐ”＞（ｍ×ｋ）／ｖの場合に、$g=2^{k*G(p'',m,k)}$について、Ｈ＝ＲＥＤＣ（Ｈ，Ｇ）ｎによる補正演算を行うステップである。ただし、Ｇは、Ｇ（ｐ”，ｍ，ｋ）＝２×ｍ－（ｖ×２ｐ”）／ｋと表され、１≦Ｇ（ｐ”，ｍ，ｋ）≦ｍ－１の範囲を満たす整数である。

【００４３】
ステップＡ３のステップＳ３０１では、第１レジスタＲＥＧ１及び第２レジスタＲＥＧ２に対し、初期値として夫々「ｎ」及び「$2^{(m-1)*k}$」を与える。なおｎの有効ワード長はｍである。
【００４４】
ステップＡ３のステップＳ３０２では、第２レジスタＲＥＧ２に対する左１ビットシフト処理と、ＲＥＧ２≧ＲＥＧ１の真偽判定と、ＲＥＧ２≧ＲＥＧ１が真である場合にＲＥＧ２－ＲＥＧ１の演算結果を第２レジスタＲＥＧ２に格納する処理とからなる２倍剰余演算をｋ＋ｖ回繰り返し、ＲＥＧ２＝H_0 ＝2^{m*k+v} （ｍｏｄ　ｎ）とする。ただし、ｖは自然数であり、（ｍ×ｋ）／ｖは整数である。
【００４５】
ステップＢ３のステップＳ３０３では、ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）ｎ として示されるＲＥＤＣ演算の結果を第２レジスタＲＥＧ２に格納する処理を、ｉ＝１，２，…，ｐ”としてｐ”回繰り返すことにより、ＲＥＧ２＝$2^{E(p",m,k)}$ （ｍｏｄ　ｎ）を計算する。ただし、Ｐ”は、$2^{p"-1}$＜（ｍ×ｋ）／ｖ≦$2^{p"}$を満たす整数であり、Ｅ（ｐ”，ｍ，ｋ）＝ｍ×ｋ＋ｖ×$2^{p"}$であり、ＲＥＤＣ（Ａ，Ｂ）ｎ は、モンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ ＝2^{-m*k}×Ａ×Ｂ（ｍｏｄ　ｎ）を示す。
【００４６】
ステップＣ３のステップＳ３０４では、$2^{p"}$＞（ｍ×ｋ）／ｖの場合に、ＲＥＤＣ（ＲＥＧ２，ｇ）ｎ として示されるＲＥＤＣ演算の結果を第２レジスタＲＥＧ２に格納する。ただし、ｇ＝$2^{k*G(p",m,k)}$ であり、Ｇ（ｐ”，ｍ，ｋ）＝２×ｍ－（ｖ×２ｐ”）／ｋである。
【００４７】
そしてステップＳ３０５では、計算した結果であるＲＥＧ２＝R^2 （ｍｏｄ　ｎ）を出力し、処理を終了する。
【００４８】
図２１は、従来法３におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図２１では、図２０を用いて示した従来法３の計算方法に必要な演算回数を演算の種類及びステップ別に示している。なお図２１において、ＳＦＴは１ビットシフトを行うシフト演算を、ＳＵＢは減算を、ＣＭＰは比較演算を、ＲＥＤＣはモンゴメリ乗算剰余演算をそれぞれ示す。
【００４９】
ステップＡ３に示される様に、従来法３では、ｑの値を用いずにH_0 の計算が行われている。またステップＳ３０３に対し、ステップＳ３０４に示す補正演算処理を追加することにより、（ｍ×ｋ）／ｖが２のべき乗の値であるという制約がなくなるので、ｖはステップＳ３０２にて示す条件にのみ従えばよい。しかも（ｍ×ｋ）／ｖの各ビット値の検出を行う必要がない。
【００５０】
次に図２１に示した図表に基づいて、従来法３における計算回数の例を説明する。
例３－１．１０２４ビットのＲＳＡ暗号の計算に適用
上記条件より、ｎは１０２４ビットである。１ワード＝３２ビットとするとｋ＝３２であり、ｎの有効ワード長ｍ＝３２となる。ｍ×ｋ＝１０２４であるので、ｖは１０２４の任意の因数（ｆａｃｔｏｒ）から選択することが可能である。Ｖ＝１の場合、ＳＦＴが３２＋１＝３３回、ＳＵＢが０．５×（３２＋１）＝１６．５回、ＣＭＰが３２＋１＝３３回、そしてＲＥＤＣがｐ＝log_2（（３２×３２）／１）＝１０回となる。
【００５１】
例３－２．１６３ビットの楕円曲線暗号の計算に適用
上記条件より、ｎは１６３ビットである。１ワード＝８ビットとするとｋ＝８であり、ｎの有効ワード長ｍ＝２１となる。ｍ×ｋ＝１６８であるので、ｖは１６８の任意の因数（ｆｏｃｔｏｒ）から選択することが可能である。ｖ＝２１の場合、ＳＦＴが８＋２１＝

２９回、ＳＵＢが０．５×（８＋２１）＝１４．５回、ＣＭＰが８＋２１＝２９回、そしてＲＥＤＣが（ｍ×ｋ）／ｖ＝（１０００）2 からｐ’－１＋Ｗ（（ｍ×ｋ）／ｖ）＝４－１＋０＝３回となる。

【００５２】

このような従来法３に示す計算方法は、例えば特許文献５に示されている。

【特許文献１】特開平８－２６３３１６号公報

【特許文献２】特開平８－３３９３１０号公報

【特許文献３】特開平１１－３０５９９５号公報

【特許文献４】米国特許第５７７７９１６号明細書

【特許文献５】国際公開第２００５／０１３２４３号パンフレット

【発明の開示】

【発明が解決しようとする課題】

【００５３】

しかしながら上述した従来法１乃至従来法３は、以下に示す様な解決すべき課題を有している。

【００５４】

課題１．

従来法１として示した計算方法は、ステップＡ１の処理において、第１レジスタＲＥＧ１に格納した「ｎ」のビット列中で上位から連続する「０」の個数を以降の計算に要するパラメータであるｑとして用いるため、データ値の最上位有効ビット（Ｍｏｓｔ　Ｓｉｇｎｉｆｉｃａｎｔ　Ｂｉｔ；以下ＭＳＢという）を算出しなければならない。ＭＳＢの算出には、ソフトウェア実装における処理効率が悪いビット単位の演算処理が必要となるという問題がある。しかも図１７に示した図表から明らかな様にシフト演算、減算及び比較演算の回数は、ｑの値に依存しており、ｑが大きい程、処理負荷が大きくなるという問題がある。このようにｑに関する処理負荷の増大という問題がある。

【００５５】

課題２．

さらに従来法１として示した計算方法は、ステップＢ１の処理において、ＲＥＤＣ演算をｐ回繰り返すことでＨ＝2^{2*m*k}（ｍｏｄ　ｎ）＝R^2（ｍｏｄ　ｎ）を計算する。このときｐは、ｐ＝$\log_2$（（ｍ×ｋ）／ｖ）を満たす整数、即ち（ｍ×ｘ）／ｖの値が２のべき乗となる値でなければならないという制約となる。この制約を満たすため、ｍ，ｋ，ｖは、ｎのビット長及び１ワードあたりのビット長からｍ及びｋを決定し、決定されたｍ及びｋに対し、（ｍ×ｋ）／ｖが２のべき乗の値となるようにｖの値を設定するという手順で決定される。即ち（ｍ×ｋ）／ｖが２のべき乗の値になるようにｖの値を設定しなければならないという制約のため、ｖの値は大きな値をとる可能性がある。図１７に示した図表から明らかな様に、シフト演算、減算及び比較演算の回数は、ｖの値に依存しており、ｖが大きい程、処理負荷が大きくなるという問題がある。このように（ｍ×ｋ）／ｖが２のべき乗の値との制約に関する処理負荷の増大という問題がある。

【００５６】

課題３．

従来法２として示した計算方法は、ステップＡ２の処理が、従来法１のステップＡ１の処理と同様であるため、従来法１と同様のｑに関する処理負荷の増大という問題がある。

【００５７】

課題４．

さらに従来法２として示した計算方法は、ステップＢ２の処理において、ＲＥＤＣ演算をｐ’－１回繰り返すために、（ｍ×ｋ）／ｖのｉ番目のビット値を検出するので、ソフトウェア実装における処理効率が悪いビット単位の演算処理が必要となるという問題がある。このようにＲＥＤＣ演算を繰り返すために、（ｍ×ｋ）／ｖの各ビット値の検出に関する問題がある。

【００５８】

課題５．

従来法３として示した計算方法は、従来法１及び従来法２に示す様にＭＳＢの算出及びｑの値に依存する処理がないという点で優れている。しかしステップＡ３の処理において、２倍剰余演算をｋ＋ｖ回繰り返すため、図２１に示した図表から明らかな様に、シフト演算、減算及び比較演算の回数は、ｋの値に依存しており、ｋが大きい程、処理負荷が大きくなる。

【００５９】

図２２は、従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図２２は、図１７に示した従来法１のステップＡ１の計算量、図１９に示した従来法２のステップＡ２の計算量及び図２０に示した従来法３のステップＡ３の計算量を示している。なお図２２において、夫々の計算方法の処理負荷の比較が容易になる様に、シフト演算ＳＦＴ、減算ＳＵＢ及び比較演算ＣＭＰの演算に要する処理負荷を同一とみなし、これらを定数ＬＣに置き換えて示している。

【００６０】

図２２に示す表より、（２．５×ｋ＋２．５×ｖ）×ＬＣ＜（５．５ｑ＋２．５ｋ＋１）×ＬＣが成立する場合、即ち（５×ｋ－２）／１１＜ｑが成立する場合、従来法３に示した計算方法は、従来法１及び従来法２より計算量が少なく効率的な方法であることが示される。しかしながらｑの値が小さく、（５×ｋ－２）＞ｑが成立する場合、即ちｑが小さい場合、従来法３に示した計算方法は、従来法１及び従来法２より計算量が多く非効率な方法であるということになる。

【００６１】

実際に良く用いられるｑの値として、例えばＲＳＡ暗号においては、ｎのビット長が２０４８、１０２４、５１２等の２のべき乗となるビット長が良く用いられており、これらの場合、ｑ＝０となる。楕円曲線暗号を用いる場合、ｎのビット長は任意の値をとることになるが、ＳＥＣＧ（Ｓｔａｎｄａｒｄｓ ｆｏｒ Ｅｆｆｅｉｃｉｅｎｔ Ｃｒｙｐｔｏｇｒａｐｈｙ Ｇｒｏｕｐ）にてＳＥＣ１として規定される規格では、１６０、１９２、２２４等の３２の倍数のビット長を推奨しており、これらのパラメータを用いる場合、いずれもｑ＝０となる。

【００６２】

従って実用上、従来法３に示した計算方法は、必ずしも従来法１及び従来法２より優れている訳ではなく、ｑの値が小さい場合、ステップＡ３の処理は、ステップＡ１及びＡ２の処理より負荷が大きいという問題がある。

【００６３】

本発明は斯かる事情に鑑みてなされたものであり、2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納し、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納し、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算することにより、従来法１乃至従来法３の問題を解決することが可能な計算方法、該計算方法を適用した計算装置、及び該計算装置を実現するためのコンピュータプログラムの提供を目的とする。

【課題を解決するための手段】

【００６４】

第１発明に係る計算方法は、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを備えた計算装置を用いて計算する計算方法において、前記計算装置は、2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納するステップと、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納する

ステップと、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算するステップとを実行することを特徴とする。

【００６５】

第２発明に係る計算方法は、第１発明において、計算した同値を用いて、べき乗剰余処理を実行することを特徴とする。

【００６６】

第３発明に係る計算装置は、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算する計算装置において、レジスタと、剰余の法ｎの負数を前記レジスタに格納する手段と、レジスタに格納されている値を桁上がり方向へ１ビットシフトする処理を、レジスタからあふれる最上位ビットが０になるまで繰り返す手段と、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算する手段とを備えることを特徴とする。

【００６７】

第４発明に係る計算装置は、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算する計算装置において、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタと、値Ａ及びＢ、並びに有効ワード長がｍである剰余の法ｎに対し、$2^{-m*k}\times A\times B$（ｍｏｄ　ｎ）として定義されるモンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ を実行する演算手段と、剰余の法ｎの負数をレジスタに格納する手段と、レジスタに格納されている値を桁上がり方向へ１ビットシフトするシフト処理を、レジスタからあふれる最上位ビットが０になるまで繰り返す手段と、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ＲＥＧ）ｎ を実行して、その結果をレジスタに格納する処理を、$2^{p-1}<m\times k\leqq 2^{p}$ を満たす整数であるｐ回繰り返す手段と、$2^{p}>m\times k$である場合に、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ｇ）ｎ を実行して、その結果をレジスタに格納する手段と（ただし$g=2^{k*G(p,m,k)}$、かつＧ（ｐ，ｍ，ｋ）＝２×ｍ－2^{p}／ｋ）、レジスタに格納されている値を、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値として出力する手段とを備えることを特徴とする。

【００６８】

第５発明に係る計算装置は、第４発明において、複数のレジスタと、ｍ個のワードを有する第１のレジスタ及びｍ個以上のワードを有する第２のレジスタに、夫々ｎ及び０を格納する手段と、第２のレジスタに格納されている値から、第１のレジスタに格納されている値を減じて、剰余の法ｎの負数を計算する手段とを更に備えることを特徴とする。

【００６９】

第６発明に係る計算装置は、第４発明において、レジスタに剰余の法ｎを格納する手段と、レジスタに格納されている値の補数を、剰余の法ｎの負数として計算する手段とを更に備えることを特徴とする。

【００７０】

第７発明に係る計算装置は、第４発明において、レジスタに剰余の法ｎを格納する手段と、レジスタに格納されている値を反転させる手段と、レジスタに格納されている値の最下位ビットを１として、剰余の法ｎの負数を計算する手段とを更に備えることを特徴とする。

【００７１】

第８発明に係る計算装置は、第４発明乃至第７発明のいずれかにおいて、前記シフト処理は、レジスタに格納されている値に該値を加算する加算処理であり、前記シフト処理によりレジスタからあふれる最上位ビットは、前記加算処理により発生したキャリー値として検出することを特徴とする。

【００７２】

第９発明に係るコンピュータプログラムは、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを備えるコンピュータに、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算させるコンピュータプログラムにおいて、コンピュータに、2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納させる手順と、コンピュータに、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納させる手順と、コンピュータに、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算させる手順とを実行させることを特徴とする。

【００７３】

第１発明、第３発明、第４発明及び第９発明では、モンゴメリ乗算剰余演算に用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値として、法ｎに関する剰余値が同じである同値を計算する。剰余値には、０以上かつ法ｎ未満でなければならないという制約があるが、同値にはその制約がない。従って剰余値ではなく同値を計算することにより、制約が緩和されるため、制約に基づく様々な処理が不要になるので、計算処理を高速化することが可能である。しかも計算結果を同値とすることにより、計算の途中で発生する中間的なデータに対しても、制約の多い剰余値だけでなく制約の少ない同値を用いることができるので、計算処理を高速化することが可能である。

【００７４】

例えば上述した従来法１のステップＡ１及び従来法２のステップＡ２では、ＲＥＧ２の値が常にｎ’未満となる様に調整しながら計算を行うため、ｑに依存した回数のシフト演算を行わなければならず、課題１及び課題３として提起した問題があった。また上述した従来法３のステップＡ３についてもＲＥＧ２の値が常にｎ未満となる様に調整しながら計算を行うため、課題５として提起した問題があった。課題１、課題３及び課題５として示したこれらの問題は、剰余値が０以上かつ法ｎ未満である値を計算するために生じたものであり、本発明では、剰余値ではなく、剰余値の同値を計算することで、これらの問題を解消し、計算処理を高速化することが可能である。

【００７５】

なお本発明では、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すという処理を行っている。シフト演算と、破棄する１ビットの値を判定する処理とを繰り返す処理は、従来法１のステップＳ１０４、従来法２のステップＳ２０４及び従来法３のステップＳ３０２にて行われるシフト演算と、比較演算とを繰り返す方法より演算効率が高い。シフト演算及び比較演算は、公開鍵暗号方式にて行われる演算に用いられる１６０～２０４８ビットの非常に長いビット長のデータに対する多ビット演算を実行するのに対し、１ビットの値の判定は、１ビット分の演算のみであるので、多ビット演算より高速な処理が可能だからである。１ビットの判定のみを用いた効率的な処理を実現することができたのは、計算の対象を剰余値に限定せずに、同値にまで拡張したことにより、様々な制約から解放されたからである。

【００７６】

また本発明では、（ｍ×ｋ）／ｖが２のべき乗の値という制約がないので、従来法１の課題２として提起した問題を解消することが可能である。

【００７７】

さらに本発明では、ＲＥＤＣ演算をｐ’－１回繰り返すために、（ｍ×ｋ）／ｖのｉ番目のビット値を検出する処理がないので、従来法２の課題４として提起した問題を解消することが可能である。

【００７８】

第２発明では、べき乗剰余処理は、同値を用いても実行することができるので、上述し

た様に処理効率が高い同値を用いて実行することにより、全体としての処理速度を向上させることが可能である。

【００７９】

第５発明乃至第８発明では、既存の演算チップを用いることができるので、実装が容易である。

【発明の効果】

【００８０】

本発明に係る計算方法、計算装置及びコンピュータプログラムは、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを用いるものであり、2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納し、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納し、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算する。

【００８１】

従来法にて使用される剰余値には、０以上かつ法ｎ未満でなければならないという制約があるが、本発明にて使用される同値にはその制約がない。従って剰余値ではなく同値を計算することにより、制約が緩和されるため、制約に基づく様々な処理が不要になるので、計算処理を高速化することが可能である等、優れた効果を奏する。しかも計算結果を同値とすることにより、計算の途中で発生する中間的なデータに対しても、制約の多い剰余値だけでなく制約の少ない同値を用いることができるので、計算処理を高速化することが可能である等、優れた効果を奏する。

【００８２】

また本発明では、同値を用いてべき乗剰余処理を実行することにより、上述した様に処理効率が高い同値を用いて実行することとなるので、全体としての処理速度を向上させることが可能である等、優れた効果を奏する。さらにＲＳＡ暗号、楕円曲線暗号等の暗号方式を用いた公開鍵暗号方式に適用することができるので、秘匿性の高い通信を高速に実現する情報セキュリティ技術を提供することが可能である等、優れた効果を奏する。

【発明を実施するための最良の形態】

【００８３】

以下、本発明をその実施の形態を示す図面に基づいて詳述する。図１は、本発明の計算装置の構成例を示すブロック図である。図１中１は、マイクロコンピュータとして機能する演算カード等の本発明の計算装置であり、計算装置１は、パーソナルコンピュータ、サーバコンピュータ等の通信装置２に組み込まれている。計算装置１は、装置全体を制御するＭＰＵ等の制御手段１１、本発明のコンピュータプログラム３等の各種コンピュータプログラム及びデータを記録したＲＯＭ、ＲＡＭ等の記録手段１２、計算に用いられる第１レジスタ１３ａ及び第２レジスタ１３ｂ、ＲＥＤＣ演算を行うコプロセッサ等の演算手段１４、並びに通信装置２とのインターフェースである接続手段１５を備えている。そして記録手段１２に記録された本発明のコンピュータプログラム３を制御手段１１により実行することにより、マイクロコンピュータとして機能する演算カードは、本発明の計算装置としての各種手順を実行する。なお第１レジスタ１３ａは、ｋビットの２進数データを格納することが可能なｍ個のワードを有するレジスタであり、第２レジスタ１３ｂは、ｍ個以上のワードを有するレジスタである。

【００８４】

本発明の計算装置１は、公開鍵暗号方式等の暗号技術を用いた通信等の処理において様々な処理を実行する。具体的には、本発明の計算装置１は、接続手段１５を介して通信装置２から受け付けた平文情報を、予め記録している公開鍵にて暗号化して暗号文を生成し、生成した暗号文を、接続手段１５を介して通信装置２へ出力する。また本発明の計算装

置１は、通信装置２が他の装置から公開鍵にて暗号化された暗号文を受信した場合に、受信した暗号文を通信装置２から接続手段１５を介して受け付け、予め記録している秘密鍵にて復号して平文を生成し、生成した平文を、接続手段１５を介して通信装置２へ出力する。なお本発明の計算装置１では、同様の技術を用いて、平文を秘密鍵にて暗号化し、暗号文を公開鍵にて復号するデジタル署名に係る処理を実行することも可能である。

【００８５】

公開鍵暗号方式の暗号技術には、ＲＳＡ暗号、楕円曲線暗号等の暗号方式が用いられる。例えばＲＳＡ暗号では、公開鍵ｅにて平文ｍを暗号化した暗号文ｃは、剰余の法ｎを用いて$c = m^e$（ｍｏｄ　ｎ）と表される。また秘密鍵ｄにて暗号文ｃを復号した平文ｍは、剰余の法ｎを用いて、$m = c^d$（ｍｏｄ　ｎ）と表される。このようにＲＳＡ暗号では、$y = a^x$（ｍｏｄ　ｎ）で表されるべき乗剰余演算処理が行われる。また楕円曲線暗号でも、乗算剰余演算処理が用いられている。

【００８６】

本発明の計算装置１は、乗算剰余演算処理の代わりに下記の式にて示す整数ａ，ｂ，ｎを用いたモンゴメリ乗算剰余と呼ばれる演算法を用いて暗号化及び復号処理を行う。

【００８７】

$y = a \times b \times R^{-1}$（ｍｏｄ　ｎ）

ただし、ｎ：剰余の法を示す整数

ａ，ｂ：$0 \leqq a, b < n$となる整数

Ｒ：2^{m*k}にて示される定数

ｋ：１ワードあたりのビット長

ｍ：ｎを表現するために必要なワードの最小個数

【００８８】

図２は、本発明の計算方法に関するモンゴメリ乗算剰余演算のアルゴリズムを示す説明図である。なお図２に示すアルゴリズムにおいて、$x = (x_{m-1}, \cdots, x_1, x_0)$は、整数値ｘを、ｍ個のワード値$x_i$（$i = m-1, \cdots, 1, 0$，$0 \leqq x_i < 2^k$）として表現する形式を示す。また図２を含む以降の図面及び以降の説明において、「：＝」とは、右辺の数値又は数式を左辺に代入することを示す。図２に示す様に夫々ｍワード値にて示されるａ，ｂ，ｎに基づいて、ｍワード値で示される値ｙを算出する場合のモンゴメリ乗算剰余演算$y = a \times b \times R^{-1}$（ｍｏｄ　ｎ）を、以降の説明では、ｙ＝ＲＥＤＣ（ａ，ｂ）ｎ 又は単にＲＥＤＣと表記する。このように定義されるＲＥＤＣは、以下に示す３つの性質を備える。

【００８９】

（性質１）法ｎは、奇数に限定される。

（性質２）値ａ，ｂが、両方ともｍ個のワード値での表現が可能な場合で、$a \times b \leqq R \times n$の条件が満たされるとき、$y = a \times b \times R^{-1}$（ｍｏｄ　ｎ）の計算が行われる。このとき$0 \leqq y < n$を満足する。

（性質３）値ａ，ｂが、両方ともｍ個のワード値での表現が可能な場合で、$a \times b \leqq R \times n$の条件が満たされないとき、$y \equiv a \times b \times R^{-1}$（ｍｏｄ　ｎ）の計算が行われる。このとき$0 \leqq y < n$を満足するとは限らない。

【００９０】

ここで性質２に示した計算$y = a \times b \times R^{-1}$（ｍｏｄ　ｎ）と、性質３に示した計算$y \equiv a \times b \times R^{-1}$（ｍｏｄ　ｎ）との差異について説明する。性質２に示した計算と性質３に示した計算との差異は、性質２に示した計算が、法ｎに関する「剰余値」を求めているのに対し、性質３に示した計算は、法ｎに関する「同値」を求めていることにある。

【００９１】

整数ｘに対して自然数の法ｎに関する除算にて計算される余りｙは、「（法ｎに関する）剰余値」と呼ばれ、$y = x$（ｍｏｄ　ｎ）として表記される。剰余値の計算において、余りｙは、０以上ｎ未満の値をとるため、上記の性質２の計算では、$0 \leqq y < n$を満足することとなる。

【００９２】
これに対し、０以上ｎ未満を満たすとは限らないが、剰余値が同一となる複数の値ｘ，ｘ’は、「（法ｎに関する）同値」と呼ばれ、ｘ’≡ｘ（ｍｏｄ　ｎ）として表記される。即ちｘの剰余値ｙ、法ｎ、及び整数ｓが、ｘ’＝ｙ＋ｓ×ｎの関係にある場合、全てのｘ’は、ｘと同値となる。例えばｎ＝５、ｘ＝１３という条件下において、ｘの法ｎに関する剰余値ｙは、ｙ＝３となる。更に同一条件下において、ｘ’＝３，８，１３，１８，２３，…の一連の値は、法ｎに関して同値であり、その剰余値は３である。このように同値は、法ｎに関して余りが同一となる一連の値であり、０以上ｎ未満に限定されない。従って上記の性質３の計算では、０≦ｙ＜ｎを満足するとは限らないこととなる。

【００９３】
上述した様にモンゴメリ乗算剰余演算は、ａ×ｂ×R^{-1}（ｍｏｄ　ｎ）であり、通常の乗算剰余演算ａ×ｂ（ｍｏｄ　ｎ）とは異なる演算を行う。よってべき乗剰余演算を正しく実行するためには、モンゴメリ乗算剰余に対して与える入力データをモンゴメリ系と呼ばれるデータに変換する必要がある。通常の乗算剰余演算に与える任意の入力データをｘ、ｘをモンゴメリ系に変換したデータをｘ’とし、ｘからｘ’への変換(モンゴメリ変換)をｘ’＝Ｍｏｎｔ（ｘ）として表し、ｘ’からｘへの変換(モンゴメリ逆変換)をｘ＝Ｍｏｎｔ$^{-1}$（ｘ’）と表した場合、これらは下記の式にて与えられる。

【００９４】
モンゴメリ変換：ｘ’＝Ｍｏｎｔ（ｘ）＝ｘ×Ｒ（ｍｏｄ　ｎ）
モンゴメリ逆変換：ｘ＝Ｍｏｎｔ$^{-1}$（ｘ’）＝ｘ’×R^{-1}（ｍｏｄ　ｎ）

【００９５】
上記式にて示したモンゴメリ変換及びモンゴメリ逆変換は、ＲＥＤＣを用いた下記の式にて示すことができる。ただしＨはＨ≡R^2（ｍｏｄ　ｎ）で表されるモンゴメリ変換パラメータと呼ばれる値であり、事前計算により求められる。

【００９６】
モンゴメリ変換：ｘ’＝ＲＥＤＣ（ｘ，Ｈ）ｎ＝ｘ×R^2×R^{-1}＝ｘ×Ｒ（ｍｏｄ　ｎ）
　ただし、Ｈ≡R^2（ｍｏｄ　ｎ）
モンゴメリ逆変換：ｘ＝ＲＥＤＣ（ｘ’，１）ｎ＝ｘ’×１×R^{-1}＝ｘ’×R^{-1}（ｍｏｄ　ｎ）

【００９７】
上述した数式に基づくモンゴメリ乗算剰余を用いたべき乗剰余演算のアルゴリズムについて説明する。図３は、本発明の計算方法に関するモンゴメリ乗算剰余演算を用いたべき乗剰余演算のアルゴリズムを示す説明図である。図３は、バイナリ法と呼ばれるべき乗剰余演算に基づくモンゴメリ乗算剰余演算のアルゴリズムを示しており、入力値ａ，ｄ，ｎからべき乗剰余演算結果ｙ＝a^d（ｍｏｄ　ｎ）を計算する。図３における１行目の処理は、ｙの初期値として１を与えることを示している。２行目の処理は、モンゴメリ変換パラメータＨ’≡R^2（ｍｏｄ　ｎ）を計算することを示している。３行目の処理は、ｙとａとに対しモンゴメリ変換を行いｙ’とａ’とを得ることを示している。４～７行目のループは、ｄのビット値に応じてモンゴメリ乗算剰余を１回又は２回繰り返す処理を、ｄの最下位ビットから最上位ビットまで繰り返すことを示している。８行目の処理は、４～７行目のループで計算されたｙ’に対し、モンゴメリ逆変換を行うことで、最終的な演算結果ｙを得ることを示している。

【００９８】
図３に示したアルゴリズムにおいて、２行目で行われるモンゴメリ変換パラメータＨ＝R^2（ｍｏｄ　ｎ）を計算する処理について説明する。図４は、本発明の計算装置１の処理を示すフローチャートである。図４は、本発明の計算装置１が、剰余の法ｎの入力を受け付け、本発明の計算処理を実行し、R^2（ｍｏｄ　ｎ）の同値であるＨ≡R^2（ｍｏｄ　ｎ）を出力する処理を示している。なお以降の説明において、ｋは、１ワードあたりのビット長を示し、ｎは、ｍワード値で示される値である。またＲ＝2^{m*k}である。なお以

降の図面及び以降の説明において、「＊」は、乗算記号「×」を示すものとする。

【００９９】

本発明のモンゴメリ変換パラメータの変換方法は、主にステップＡ、ステップＢ及びステップＣにて構成される。ステップＡは、2^{m*k+1} の法ｎに関する同値$H_0 \equiv 2^{m*k+1}$（ｍｏｄ　ｎ）を計算するステップである。ステップＢは、ＲＥＤＣ演算により、H_0 から$2^{E(p,m,k)}$（ｍｏｄ　ｎ）の同値$H \equiv 2^{E(p,m,k)}$（ｍｏｄ　ｎ）を計算するステップである。ただし、Ｐは、$2^{p-1} < m \times k \leqq 2^p$ を満たす整数であり、Ｅ（Ｐ，ｍ，ｋ）＝ｍ×ｋ＋2^p である。ステップＣは、$2^p > m \times k$ の場合に、$g = 2^{k*G(p,m,k)}$について、Ｈ＝ＲＥＤＣ（Ｈ，Ｇ）ｎ による補正演算を行うステップである。ただし、Ｇは、Ｇ（ｐ，ｍ，ｋ）＝２×ｍ－2^p ／ｋと表され、１≦Ｇ（ｐ，ｍ，ｋ）≦ｍ－１の範囲を満たす整数である。

【０１００】

本発明の計算装置１は、ステップＡにおけるステップＳ１の処理として、第１レジスタ１３ａ及び第２レジスタ１３ｂに対し、初期値として夫々「ｎ」及び「０」を与える初期化を行う。ただし、ｎの有効ワード長は、ｍである。

【０１０１】

図５は、本発明の計算装置１が備える第１レジスタ１３ａ及び第２レジスタ１３ｂに格納される値を概念的に示す説明図である。図５中、ＲＥＧ１は、第１レジスタ１３ａに格納されている値を示し、ＲＥＧ２は、第２レジスタ１３ｂに格納されている値を示している。図５は、ステップＡにおけるステップＳ１の処理が行われた状態を示しており、第１レジスタ１３ａには、初期値としてｎが格納されており、第２レジスタ１３ｂには、初期値として０が格納されている。

【０１０２】

図４に示すフローチャートに戻り、本発明の計算装置１は、ステップＡにおけるステップＳ２の処理として、2^{m*k} と法ｎに関する同値$REG2 \equiv 2^{m*k}$（ｍｏｄ　ｎ）を計算する。ステップＡにおけるステップＳ２の処理は、第２レジスタ１３ｂに格納されている値から第１レジスタ１３ａに格納されている値を減じて、得られた結果である法ｎの負数を第２レジスタ１３ｂに格納する処理により行われる。

【０１０３】

第２レジスタ１３ｂに格納されている値から第１レジスタ１３ａに格納されている値を減じた結果、即ちＲＥＧ２－ＲＥＧ１＝０－ｎは、整数ｓを用いて2^{m*k} ＋ｓ×ｎという形で示すことができるので、2^{m*k} と法ｎに関して同値であり、しかもｍワードで示すことが可能な値である。

【０１０４】

なおステップＡにおけるステップＳ２の処理は、演算処理（ＲＥＧ２：＝ＲＥＧ２－ＲＥＧ１）を行うのではなく、第１レジスタ１３ａに格納されている値ｎに関する２の補数値を求め、求められた２の補数値を第２レジスタ１３ｂに格納するようにしてもよい。値ｎに関する２の補数値は、第１レジスタ１３ａに格納されているｎの全ビット値を反転させ、更に第１レジスタ１３ａに格納されている値の最下位ビットに１を加算することにより求められる。

【０１０５】

図６は、本発明の計算装置１が備える第１レジスタ１３ａ及び第２レジスタ１３ｂに格納される値を概念的に示す説明図である。図６は、ステップＡにおけるステップＳ２の処理が行われた状態を示しており、第１レジスタ１３ａには、初期値として格納されたｎが格納されており、第２レジスタ１３ｂには、０－ｎとして計算された2^{m*k} の法ｎに関する同値が格納されている。

【０１０６】

図４に示すフローチャートに戻り、本発明の計算装置１は、ステップＡにおけるステップＳ３の処理として、2^{m*k+1} と法ｎに関する同値$REG2 = 2^{m*k+1}$（ｍｏｄ　ｎ）を計算する。ステップＡにおけるステップＳ３の処理は、更に詳細には、第２レジスタ１３

ｂに格納されている値を左１ビットシフト演算する処理（ステップＳ３－１）と、左１ビットシフト演算によりあふれた値、即ち演算前の最上位ビット値を判定する処理（ステップＳ３－２）とを含む。ステップＳ３－１の左１ビットシフト演算する処理とは、第２レジスタ１３ｂの各桁の値を繰り上げる処理、即ち第２レジスタ１３ｂに格納されている値を２倍して、最上位の桁のビット値をあふれた値として破棄する処理である。そしてステップＳ３－２において、あふれた値が「１」であると判定した場合、第２レジスタ１３ｂに格納されている値は、2^{m*k} と法ｎに関して同値の値であると判断し、ステップＳ３－１へ戻り、以降の処理を繰り返す。またあふれた値が「０」であると判定した場合、第２レジスタ１３ｂに格納されている値は、2^{m*k+1} と法ｎに関して同値の値であると判断し、ステップＳ３の処理を終了する。

【０１０７】

なおステップＡにおけるステップＳ３のステップＳ３－１の処理は、第２レジスタ１３ｂに格納されている値に、第２レジスタ１３ｂに格納されている値を加算する処理、即ち演算処理（ＲＥＧ２：＝ＲＥＧ２＋ＲＥＧ２）を行う処理に代替することも可能である。またステップＳ３－２の処理は、ステップＳ３－１の処理により、キャリー値の発生の有無を判定する処理に代替することも可能である。その場合、キャリー値が発生したと判定した場合には、ステップＳ３－１へ戻り、キャリー値が発生していないと判定した場合、ステップＳ３の処理を終了する。

【０１０８】

図７は、本発明の計算装置１が備える第２レジスタ１３ｂに格納される値を概念的に示す説明図である。なお図７中破線で示した四角形内に示された数値は、左１ビットシフト演算によりあふれた値を示す。図７（ａ）は、ステップＡにおけるステップＳ３の処理を実行する前の状態を示しており、$ＲＥＧ２ \equiv 2^{m*k}$ （ｍｏｄ　ｎ）である。図７（ｂ）は、図７（ａ）の状態からステップＳ３－１にて左１ビットシフト演算処理を１回実行した状態を示しており、$ＲＥＧ２ \equiv 2^{m*k}$ （ｍｏｄ　ｎ）である。図７（ｂ）に示す様にあふれた値は「１」であるので、ステップＳ３－１へ戻り、再度、左１ビットシフト演算処理を実行する。２度目の左１ビットシフト演算処理を実行した状態が図７（ｃ）であり、$ＲＥＧ２ \equiv 2^{m*k}$ （ｍｏｄ　ｎ）である。図７（ｃ）に示す様にあふれた値は、「１」であるので、ステップＳ３０１へ戻り、再度、左１ビットシフト演算処理を実行する。３度目の左１ビットシフト演算処理を実行した状態が図７（ｄ）である。図７（ｄ）に示す様にあふれた値は、「０」であるので、$ＲＥＧ２ \equiv 2^{m*k+1}$ （ｍｏｄ　ｎ）であると判断し、ステップＳ３の処理を終了する。このようにステップＡにおけるステップＳ３では、あふれたビット値を切り捨てながら左１ビットシフト演算を繰り返すことで、結果を常にｍワードの範囲としながらも、ステップＳ３の終了時には、2^{m*k+1} との同値を計算することができる。

【０１０９】

ステップＡにおけるステップＳ３の処理により、2^{m*k+1} との同値を計算することができる理由を説明する。図８及び図９は、本発明の計算装置１が備える第２レジスタ１３ｂに格納される値を概念的に示す説明図である。図８は、ステップＡにおけるステップＳ３の処理を実行することによりあふれる値が「０」である場合を示しており、図８（ａ）が、左１ビットシフト演算処理を実行する前の状態であり、図８（ｂ）が、左１ビットシフト演算処理を実行した後の状態を示している。また図９は、ステップＡにおけるステップＳ３の処理を実行することによりあふれる値が「１」である場合を示しており、図９（ａ）が、左１ビットシフト演算処理を実行する前の状態であり、図９（ｂ）が、左１ビットシフト演算処理を実行した後の状態を示している。

【０１１０】

図８（ｂ）及び図９（ｂ）に示す様に、左１ビットシフト演算処理を実行した直後のあふれた１ビットを含むｍ×ｋ＋１ビットの値は、2^{m*k} （ｍｏｄ　ｎ）と同値である図８（ａ）及び図９（ａ）に示す値を２倍した値であるので、2^{m*k+1} （ｍｏｄ　ｎ）と同値である。図８に示す様にあふれた最上位ビットの値が「０」である場合、あふれた最上位

ビットの切り捨てにより実質的な値が変わることはないので、第２レジスタ１３ｂに格納されている値は、2^{m*k}（ｍｏｄ　ｎ）を２倍した2^{m*k+1}（ｍｏｄ　ｎ）と同値である。図９に示す様にあふれた最上位ビットの値が「１」である場合、最上位ビットの切り捨ては、2^{m*k}（ｍｏｄ　ｎ）を２倍した2^{m*k+1}（ｍｏｄ　ｎ）からの2^{m*k}の減算であり、その結果は、2^{m*k}（ｍｏｄ　ｎ）と同値である。即ちＲＥＧ２≡２×2^{m*k}（ｍｏｄ　ｎ）－2^{m*k}（ｍｏｄ　ｎ）≡2^{m*k}（ｍｏｄ　ｎ）である。

【０１１１】

このように本発明の計算装置１のステップＡでは、ステップＳ３にて、左１ビットシフト演算処理及びあふれたビット値に対する判定処理を繰り返すことにより、2^{m*k+1}と法ｎに関する同値である$H_0 \equiv 2^{m*k+1}$（ｍｏｄ　ｎ）の値を計算することができる。

【０１１２】

図４に示すフローチャートに戻り、本発明の計算装置１は、ステップＢにおけるステップＳ４の処理として、演算手段１４により、ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）ｎとして示されるＲＥＤＣ演算を実行し、その結果を第２レジスタ１３ｂに格納する処理を、ｉ＝１，２，…，ｐとしてｐ回繰り返すことにより、ＲＥＧ２＝$2^{E(p,m,k)}$（ｍｏｄ　ｎ）を計算する。ただし、Ｐは、$2^{p-1}<m\times k\leqq 2^{p}$を満たす整数であり、Ｅ（Ｐ，ｍ，ｋ）＝ｍ×ｋ＋$2^{p}$であり、ＲＥＤＣ（Ａ，Ｂ）ｎは、モンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ＝2^{-m*k}×Ａ×Ｂ（ｍｏｄ　ｎ）を示す。

【０１１３】

本発明の計算装置１は、ステップＣにおけるステップＳ５の処理として、$2^{p}>m\times k$の真偽を判定し、$2^{p}>m\times k$が真であると判定した場合に、演算手段１４により、ＲＥＤＣ（ＲＥＧ２，ｇ）ｎとして示されるＲＥＤＣ演算を実行する補正演算を行い、その結果を第２レジスタ１３ｂに格納する。なお$2^{p}>m\times k$が偽であると判定した場合、演算手段１４によるＲＥＤＣ演算を実行する補正演算は行われない。ただし、ｇ＝$2^{k*G(p,m,k)}$であり、Ｇ（ｐ，ｍ，ｋ）＝２×ｍ－2^{p}／ｋである。

【０１１４】

本発明の計算装置１は、ステップＳ６の処理として、第２レジスタ１３ｂに格納されている計算結果、即ちＲＥＧ２$\underline{\equiv}R^2$（ｍｏｄ　ｎ）を出力し、処理を終了する。そして本発明の計算装置１は、出力された結果であるR^2（ｍｏｄ　ｎ）を用いてべき乗剰余処理を実行し、更に暗号化及び／又は復号を行う。

【０１１５】

図１０は、本発明のモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図１０では、図４を用いて示した本発明のモンゴメリ変換パラメータの計算方法に要する演算回数を演算の種類及びステップ別に示している。なお図１０において、ＳＦＴは１ビットシフトを行うシフト演算を、ＳＵＢは減算を、ＣＰＬは２の補数計算を、ＢＩＴＣＨＫは１ビット値の検出計算を、ＲＥＤＣはモンゴメリ乗算剰余演算をそれぞれ示す。また図１０において、ｑは、ｎの最上位から連続する「０」の個数を示す。

【０１１６】

次に図１０に示した図表に基づいて、本発明の計算方法の演算回数の例を説明する。

【実施例１】

【０１１７】

１０２４ビットのＲＳＡ暗号（１ワードが３２ビット：ｋ＝３２）の計算に適用

１０２４ビットの法ｎとして、ｎ＝2^{1023}＋１の場合の実施例について説明する。ＲＳＡ暗号で用いられる数値は、ｎが2つの素数ｐ，ｑの積であるという条件があり、実施例１におけるｎはこの条件を満たさない。しかし本発明の計算方法は、法ｎが任意の奇数値である場合に、2^{2*m*k}の法nに関する同値Ｈ≡2^{2*m*k}（ｍｏｄ　ｎ）を計算する方法であり、ｎが素数の積であることに限定しない。よって本実施例のｎはＲＳＡ暗号の条件は満たさないが、本発明の計算方法に係る条件は満たすものであり、しかも非常に簡単な形で表現できる値であるので、本発明の実施例１の理解を容易にするものと考えられる。以上を踏まえた上で、１０２４ビットのｎとして、ｎ＝2^{1023}＋１の場合の実施例につい

て説明する。

【０１１８】

表題の条件に示す様に１ワードが３２ビットであることから、１０２４ビットは３２ワードで示されるのでｍ＝３２となる。$H_0 \equiv 2^{2*m*k}$（ｍｏｄ　ｎ）$\equiv 2^{2048}$（ｍｏｄ　ｎ）を計算する場合、必要な計算量は、図１０より、ＳＦＴが１回、ＳＵＢ（ＣＰＬ）が１回、ＢＩＴＣＨＫが１回、そしてＲＥＤＣが１０回となる。具体的な計算を以下に示す。

【０１１９】

ステップＡにおけるステップＳ１

ＲＥＧ１：＝ｎ＝$（１００\cdots０１）_{2,1024}$

ＲＥＧ２：＝０

第１レジスタ１３ａ及び第２レジスタ１３ｂを初期化する。ただし、ａ＝$（ｂ）_{2,c}$は、数値ａをｃビットの２進数表現した結果がｂであることを示す。

【０１２０】

ステップＡにおけるステップＳ２

ＲＥＧ２：＝０－ｎ＝$（０１１１\cdots１１）_{2,1024}$

なおＲＥＧ２：＝（ＲＥＧ１の２の補数）とすることによっても同じ結果を得ることが可能である。またＲＥＧ１の全ビットを反転し、更にその最下位ビットに１をセットする様にしてもよい。

【０１２１】

ステップＡにおけるステップＳ３

ＲＥＧ２＝$（０１１１\cdots１１）_{2,1024}$を左１ビットシフト演算し、ＲＥＧ２＝$（１１１\cdots１１０）_{2,1024}$とする。そしてあふれた値が「０」であると判定し、ステップＳ４へ進む。なおこのときＲＥＧ２（ｍｏｄ　ｎ）＝$（１１１\cdots１１０）_{2,1024}$（ｍｏｄ　ｎ）＝$（０１１\cdots１１００）_{2,1024}$及び2^{1025}（ｍｏｄ　ｎ）＝$（０１１\cdots１１００）_{2,1024}$より、演算結果の正しさが実証される。

【０１２２】

ステップＢにおけるステップＳ４

ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）

上記処理を$2^9 <ｍ\times ｋ＝１０２４\leqq 2^{10}$より決定されるｐ＝１０回繰り返す。

１回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）

$\equiv 2^{1024+1}\times 2^{1024+1}\times 2^{-1024} \equiv 2^{1024+2}$（ｍｏｄ　ｎ）

２回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）

$\equiv 2^{1024+2}\times 2^{1024+2}\times 2^{-1024} \equiv 2^{1024+4}$（ｍｏｄ　ｎ）

３回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）

$\equiv 2^{1024+2}\times 2^{1024+2}\times 2^{-1024} \equiv 2^{1024+8}$（ｍｏｄ　ｎ）

：　　：

９回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）

$\equiv 2^{1024+256}\times 2^{1024+256}\times 2^{-1024} \equiv 2^{1024+512}$（ｍｏｄ　ｎ）

１０回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）

$\equiv 2^{1024+512}\times 2^{1024+512}\times 2^{-1024} \equiv 2^{1024+1024}$（ｍｏｄ　ｎ）

上記計算よりＲＥＧ２$\equiv 2^{2048}$（ｍｏｄ　ｎ）を得る。

【０１２３】

ステップＣにおけるステップＳ５

2^p（$＝2^{10}$）$>ｍ\times ｋ$（＝１０２４）が偽であるので、補正演算は実行しない。

【０１２４】

ステップＳ６

ＲＥＧ２$\equiv H_0 \equiv 2^{2048}$（ｍｏｄ　ｎ）を出力し、処理を終了する。

【実施例２】

【０１２５】

１６３ビットの楕円曲線暗号（１ワードが８ビット：ｋ＝８）の計算に適用
１６３ビットの法ｎとして、ｎ＝0x7,0263d95a,880adfbc,e3c1648d,44ce22fa,813980fbの場合の実施例について説明する。ただし上記の0x…は、１６進数で表現された数値を示す。１ワードが８ビットであることから、１６３ビットは２１ワードで示されるのでｍ＝２１となる。H0 ≡2^{2*m*k} （ｍｏｄ　ｎ）≡2^{326} （ｍｏｄ　ｎ）を計算する場合、必要な計算量は、図１０より、ＳＦＴが６回、ＳＵＢ（ＣＰＬ）が１回、ＢＩＴＣＨＫが６回、そしてＲＥＤＣが８回となる。具体的な計算を以下に示す。

【０１２６】
ステップＡにおけるステップＳ１
ＲＥＧ１：＝ｎ＝0x7,0263d95a,880adfbc,e3c1648d,44ce22fa,813980fb
ＲＥＧ２：＝０

【０１２７】
ステップＡにおけるステップＳ２
ＲＥＧ２：＝０－ｎ＝0xf8,fd9c26a5,77f52043,1c3e9b72,bb31dd05,7ec67f05
なおＲＥＧ２：＝（ＲＥＧ１の２の補数）も同様である。

【０１２８】
ステップＡにおけるステップＳ３
ＲＥＧ２＝0xf8,fd9c26a5,77f52043,1c3e9b72,bb31dd05,7ec67f05 を左１ビットシフト演算し、ＲＥＧ２＝0xf1,fb384d4a,efea4086,387d36e5,7663ba0a,fd8cfe0a とする。このときあふれた値が「１」であると判定し、同様の処理を繰り返す。即ち２回目の処理として、
ＲＥＧ２＝0xf1,fb384d4a,efea4086,387d36e5,7663ba0a,fd8cfe0a を左１ビットシフト演算し、ＲＥＧ２＝0xe3,f6709a95,dfd4810c,70fa6dca,ecc77415,fb19fc14 とする。このときあふれた値が「１」であると判定し、同様の処理を繰り返す。即ち３回目の処理として、
ＲＥＧ２＝0xe3,f6709a95,dfd4810c,70fa6dca,ecc77415,fb19fc14 を左１ビットシフト演算し、ＲＥＧ２＝0xc7,ece1352b,bfa90218,e1f4db95,d98ee82b,f633f828 とする。このときあふれた値が「１」であると判定し、同様の処理を繰り返す。即ち４回目の処理として、
ＲＥＧ２＝0xc7,ece1352b,bfa90218,e1f4db95,d98ee82b,f633f828 を左１ビットシフト演算し、ＲＥＧ２＝0x8f,d9c26a57,7f520431,c3e9b72b,b31dd057,ec67f050 とする。このときあふれた値が「１」であると判定し、同様の処理を繰り返す。即ち５回目の処理として、
ＲＥＧ２＝0x8f,d9c26a57,7f520431,c3e9b72b,b31dd057,ec67f050 を左１ビットシフト演算し、ＲＥＧ２＝0x1f,b384d4ae,fea40863,87d36e57,663ba0af,d8cfe0a0 とする。このときあふれた値が「１」であると判定し、同様の処理を繰り返す。即ち６回目の処理として、
ＲＥＧ２＝0x1f,b384d4ae,fea40863,87d36e57,663ba0af,d8cfe0a0 を左１ビットシフト演算し、ＲＥＧ２＝0x3f,6709a95d,fd4810c7,0fa6dcae,cc77415f,b19fc140 とする。このときあふれた値が「０」であると判定し、ステップＳ４へ進む。なおこのときＲＥＧ２（ｍｏｄ　ｎ）＝2^{169} （ｍｏｄ　ｎ）＝0x5187052f,34e63323,0dda53b7,61380691,269a386dより、演算結果の正しさが実証される。

【０１２９】
ステップＢにおけるステップＳ４
ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
上記処理を2^{7} ＜ｍ×ｋ＝１０２４≦2^{8} より決定されるｐ＝８回繰り返す。
１回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
≡2^{168+1} ×2^{168+1} ×2^{-168}≡2^{168+2} （ｍｏｄ　ｎ）
２回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
≡2^{168+2} ×2^{168+2} ×2^{-168}≡2^{168+4} （ｍｏｄ　ｎ）

３回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{168+4}\times 2^{168+4}\times 2^{-168}\equiv 2^{168+8}$（ｍｏｄ　ｎ）
：　　：
７回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{168+64}\times 2^{168+64}\times 2^{-168}\equiv 2^{168+128}$（ｍｏｄ　ｎ）
８回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{168+128}\times 2^{168+128}\times 2^{-168}\equiv 2^{168+256}$（ｍｏｄ　ｎ）
上記計算よりＲＥＧ２≡2^{424}（ｍｏｄ　ｎ）

【０１３０】

ステップＣにおけるステップＳ５
2^{p}（＝2^{8}）＞ｍ×ｋ（＝１６８）が真であるので、補正演算を実行する。
補正演算
ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ｇ）≡$2^{424}\times 2^{80}\times 2^{-168}\equiv 2^{336}$
なお上記の計算において、
Ｇ（ｐ，ｍ，ｋ）＝２×ｍ－（2^{p}／ｋ）
Ｇ（８，２１，８）＝２×２１－（2^{8}／８）＝１０
さらに、
ｇ＝$2^{k*G(p,m,k)}=2^{8*10}=2^{80}$
以上により決定されるｇ＝2^{80}を用いて補正演算が実行される。

【０１３１】

ステップＳ６
ＲＥＧ２≡Ｈ≡2^{336}（ｍｏｄ　ｎ）を出力し、処理を終了する。

【実施例３】

【０１３２】

１６０ビットの楕円曲線暗号（１ワードが３２ビット：ｋ＝３２）の計算に適用
１６０ビットの法ｎとして、ｎ＝0x89381a5a,0ff02e5e,42d13b94,b6e022e6,96f53721の場合の実施例について説明する。ただし上記の0x…は、１６進数で表現された数値を示す。１ワードが３２ビットであることから、１６０ビットは５ワードで示されるのでｍ＝５となる。そしてＨ≡2^{2*m*k}（ｍｏｄ　ｎ）≡2^{320}（ｍｏｄ　ｎ）を計算する場合、必要な計算量は、図１０より、ＳＦＴが１回、ＳＵＢ（ＣＰＬ）が１回、ＢＩＴＣＨＫが１回、そしてＲＥＤＣが８回となる。具体的な計算を以下に示す。

【０１３３】

ステップＡにおけるステップＳ１
ＲＥＧ１：＝ｎ＝0x89381a5a,0ff02e5e,42d13b94,b6e022e6,96f53721
ＲＥＧ２：＝０

【０１３４】

ステップＡにおけるステップＳ２
ＲＥＧ２：＝０－ｎ＝0x76c7e5a5,f00fd1a1,bd2ec46b,491fdd19,690ac8df
なおＲＥＧ２：＝（ＲＥＧ１の２の補数）も同様である。

【０１３５】

ステップＡにおけるステップＳ３
ＲＥＧ２＝0x76c7e5a5,f00fd1a1,bd2ec46b,491fdd19,690ac8dfを左１ビットシフト演算子、ＲＥＧ２＝0xed8fcb4b,e01fa343,7a5d88d6,923fba32,d21591beとする。そしてあふれた値が「０」であると判定し、ステップＳ４へ進む。なおこのときＲＥＧ２（ｍｏｄ　ｎ）＝2^{161}（ｍｏｄ　ｎ）＝0x6457b0f1,d02f74e5,378c4d41,db5f974c,3b205a9dより、演算結果の正しさが実証される。

【０１３６】

ステップＢにおけるステップＳ４
ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
上記処理を2^{7}＜ｍ×ｋ＝１０２４≦2^{8}より決定されるｐ＝８回繰り返す。

1回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{160+1}\times 2^{160+1}\times 2^{-160}\equiv 2^{160+2}$（ｍｏｄ　ｎ）
2回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{160+2}\times 2^{160+2}\times 2^{-160}\equiv 2^{160+4}$（ｍｏｄ　ｎ）
3回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{160+4}\times 2^{160+4}\times 2^{-160}\equiv 2^{160+8}$（ｍｏｄ　ｎ）
：　　：
7回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{160+64}\times 2^{160+64}\times 2^{-160}\equiv 2^{160+128}$（ｍｏｄ　ｎ）
8回目　ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ＲＥＧ２）
　　≡$2^{160+128}\times 2^{160+128}\times 2^{-160}\equiv 2^{160+256}$（ｍｏｄ　ｎ）
上記計算よりＲＥＧ２≡2^{416}（ｍｏｄ　ｎ）

【０１３７】
ステップＣにおけるステップＳ５
2^{p}（＝2^{8}）＞ｍ×ｋ（＝１６０）が真であるので、補正演算を実行する。
補正演算
ＲＥＧ２：＝ＲＥＤＣ（ＲＥＧ２，ｇ）≡$2^{416}\times 2^{64}\times 2^{-160}\equiv 2^{320}$
なお上記の計算において、
Ｇ（ｐ，ｍ，ｋ）＝２×ｍ－（2^{p}／ｋ）
Ｇ（８，５，３２）＝２×５－（2^{8}／３２）＝２
さらに、
ｇ＝$2^{k*G(p,m,k)}=2^{32*2}=2^{64}$
以上により決定されるｇ＝2^{64}を用いて補正演算が実行される。

【０１３８】
ステップＳ６
ＲＥＧ２≡Ｈ≡2^{320}（ｍｏｄ　ｎ）を出力し、処理を終了する。

【０１３９】
次に本発明の計算方法と、前述した従来法１乃至従来法３との演算回数を比較する。図１１は、本発明の計算方法及び従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図１１は、図１０に示した本発明の計算方法のステップＡ、ステップＢ及びステップＣの計算量、図１７に示した従来法１のステップＡ１及びステップＢ１の計算量、図１９に示した従来法２のステップＡ２及びステップＢ２の計算量、並びに図２１に示した従来法３のステップＡ３、ステップＢ３及びステップＣ３の計算量を示している。なお図１１において、夫々の計算方法の処理負荷の比較が容易になる様に、多ビット演算であるシフト演算ＳＦＴ、減算ＳＵＢ、２の補数計算ＣＰＬ及び比較演算ＣＭＰの演算に要する処理負荷を同一と見なし、これらを定数ＬＣに置き換えて示している。また１ビット値の検出計算ＢＩＴＣＨＫの計算量を定数ＳＣとして示し、モンゴメリ乗算剰余演算ＲＥＤＣの計算量を定数ＲＥＤＣとして示している。なお１ビット演算であるＢＩＴＣＨＫは、多ビット演算より計算量が小さいため、ＢＩＴＣＨＫ＜ＬＣ，ＲＥＤＣが成立するものとする。図１１において、ＬＣ及びＳＣとして示した列は、従来法１のステップＡ１、従来法２のステップＡ２、従来法３のステップＡ３及び本発明の計算方法におけるステップＡに係る計算量を示している。またＲＥＤＣとして示した列は、従来法１のステップＢ１、従来法２のステップＢ２、従来法３のステップＢ３及びステップＣ３並びに本発明の計算方法におけるステップＢ及びステップＣに係る計算量を示している。

【０１４０】
まず従来法１のステップＡ１、従来法２のステップＡ２及び従来法３のステップＡ３と、本発明の計算方法におけるステップＡとに係る計算量を比較する。従来法１のステップＡ１又は従来法２のステップＡ２の計算量と、本発明の計算方法におけるステップＡの計算量との差を以下に計算する。

【０１４１】
（従来法１又は従来法２の計算量）－（本発明の計算方法における計算量）
＝（５．５ｑ＋２．５ｖ＋１）×ＬＣ－（（ｑ＋２）×ＬＣ＋（ｑ＋１）×ＳＣ）
＝（４．５ｑ＋２．５ｖ－１）×ＬＣ－（ｑ＋１）×ＳＣ
＝（３．５ｑ＋２．５ｖ－２）×ＬＣ＋（ｑ＋１）×（ＬＣ－ＳＣ）
【０１４２】
上記計算において、ｑ≧０、ｖ≧１及びＬＣ＞ＳＣであるから、計算結果は正の値をとる。従って従来法１又は従来法２の計算量より、本発明の計算方法における計算量の方が小さいことが証明される。
【０１４３】
従来法３のステップＡ３の計算量と、本発明の計算方法におけるステップＡの計算量との差を以下に計算する。
【０１４４】
（従来法３の計算量）－（本発明の計算方法における計算量）
＝（２．５ｋ＋２．５ｖ＋１）×ＬＣ－（（ｑ＋２）×ＬＣ＋（ｑ＋１）×ＳＣ）
＝（２．５ｋ＋２．５ｖ－ｑ－１）×ＬＣ－（ｑ＋１）×ＳＣ
＝（２．５ｋ＋２．５ｖ－２ｑ－２）×ＬＣ＋（ｑ＋１）×（ＬＣ－ＳＣ）
【０１４５】
上記計算結果中、第２項は、ｑ≧０及びＬＣ＞ＳＣであるから正の値をとる。また上記計算結果の第１項のＬＣの係数は、ｖ≧１より以下の様に示すことができる。
【０１４６】
２．５ｋ＋２．５ｖ－２ｑ－２≧２．５×（ｋ－ｑ）＋０．５ｑ＋０．５
【０１４７】
上記不等式において、最上位から連続する「０」の個数ｑは、１ワードあたりのビット長未満となることは明らかであるので、ｑ≧０及びｑ＜ｋが成立する。従って上記不等式は以下の様になり、第１項は正の値をとる。
【０１４８】
２．５ｋ＋２．５ｖ－２ｑ－２≧２．５×（ｋ－ｑ）＋０．５ｑ＋０．５＞０
【０１４９】
以上の計算結果より、従来法３の計算量より、本発明の計算方法における計算量の方が小さいことが証明される。従って従来法１のステップＡ１、従来法２のステップＡ２、従来法３のステップＡ３及び本発明の計算方法におけるステップＡとに係る計算量を比較した場合、本発明の計算方法におけるステップＡに係る計算量が最も小さく本発明の計算方法が優れていることが証明される。
【０１５０】
次に従来法１のステップＢ１、従来法２のステップＢ２、従来法３のステップＢ３及びステップＣ３と、本発明の計算方法におけるステップＢ及びステップＣとに係る計算量とを加味して全体の計算量を、実施例を挙げて比較する。ＲＥＤＣの計算量は、条件によって様々に変化する。上述した様に本発明の計算装置１は、コプロセッサを用いた演算手段１４により、ＲＥＤＣ演算を行うため、高速な計算を実現している。従ってここではＲＥＤＣの計算量がＬＣの計算量と同等であるという前提に基づいて、計算量の比較を行う。またｖの値は、夫々の実施例及び計算方法において、可及的最小値を選択するものとする。ｖが最小の値を選択する理由は、ＬＣ及びＳＣの回数はｖに比例して大きくなるのに対し、ＲＥＤＣの回数は$\log_2(1/v)$に比例して減少するからである。例えばｖの値を２倍に増加させた場合、ＬＣ及びＳＣの回数も２倍されるのに対し、ＲＥＤＣは1回しか減少しない。またＬＣ＝ＲＥＤＣという条件では、ＬＣ及びＳＣの回数並びにＲＥＤＣの回数の合計がそのまま全体の計算量となることも合わせて考慮すると、最小のｖの値を選択することで全体の計算量が最小になると考えられる。なお以下に示す実施例は、実施例４が前述した実施例１に対応し、実施例５が前述した実施例２に対応し、そして実施例６が前述した実施例３に対応する。

【実施例４】
【０１５１】
１０２４ビットのＲＳＡ暗号（１ワードが３２ビット：ｋ＝３２）の計算に適用
１ワードが３２ビットの場合、１０２４ビットは３２ワードで示される。従ってｋ＝３２，ｍ＝３２となる。また最上位から連続する「０」の個数を示すｑ＝０である。
【０１５２】
従来法１．
ｍ×ｋ＝１０２４より、（ｍ×ｋ）／ｖが２のべき乗値となるためのｖの最小値は１であるので、ｖ＝１を選択する。
ステップＡ１．
（５．５ｑ＋２．５ｖ＋１）×ＬＣ＝３．５×ＬＣ
ステップＢ１．
ｐ×ＲＥＤＣ＝ｌｏｇ$_2$（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝１０×ＲＥＤＣ
合計
３．５×ＬＣ＋１０×ＲＥＤＣ＝１３．５×ＬＣ
【０１５３】
従来法２．
（ｍ×ｋ）／ｖが２のべき乗値である必要はないので、ｖ＝１を選択する。
ステップＡ２．
（５．５ｑ＋２．５ｖ＋１）×ＬＣ＝３．５×ＬＣ
ステップＢ２．
ｐ’－１＋Ｗ（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝（１１－１＋Ｗ（（１０００００００００）2,11））×ＲＥＤＣ＝１０×ＲＥＤＣ
合計
３．５×ＬＣ＋１０×ＲＥＤＣ＝１３．５×ＬＣ
【０１５４】
従来法３．
（ｍ×ｋ）／ｖが２のべき乗値である必要はないので、ｖ＝１を選択する。
ステップＡ３．
（２．５ｋ＋２．５ｖ）×ＬＣ＝８２．５×ＬＣ
ステップＢ３及びステップＣ３
ｐ”×ＲＥＤＣ＝ｌｏｇ$_2$（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝１０×ＲＥＤＣ
なお（ｍ×ｋ）／ｖが２のべき乗値をとるため、補正演算は行わない。
合計
８２．５×ＬＣ＋１０×ＲＥＤＣ＝９２．５×ＬＣ
【０１５５】
本発明の計算方法
ステップＡ
（ｑ＋１）×ＬＣ＋（ｑ＋１）×ＳＣ＝ＬＣ＋ＳＣ
ステップＢ及びステップＣ
ｐ×ＲＥＤＣ＝ｌｏｇ$_2$（ｍ×ｋ）×ＲＥＤＣ＝１０×ＲＥＤＣ
なおｍ×ｋが２のべき乗値をとるため、補正演算は行わない。
合計
ＬＣ＋ＳＣ＋１０×ＲＥＤＣ＝１１×ＬＣ＋ＳＣ
【実施例５】
【０１５６】
１６３ビットの楕円曲線暗号（１ワードが８ビット：ｋ＝８）の計算に適用
１ワードが８ビットの場合、１６３ビットは２１ワードで示される。従ってｋ＝８，ｍ＝２１となる。また最上位から連続する「０」の個数を示すｑ＝５である。
【０１５７】

従来法１．
　ｍ×ｋ＝１６８より、（ｍ×ｋ）／ｖが２のべき乗値となるためのｖの最小値は２１であるので、ｖ＝２１を選択する。
ステップＡ１．
　（５．５ｑ＋２．５ｖ＋１）×ＬＣ＝（２７．５＋５２．５＋１）×ＬＣ＝８１×ＬＣ
ステップＢ１
　ｐ×ＲＥＤＣ＝ｌｏｇ$_2$（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝３×ＲＥＤＣ
合計
　８１×ＬＣ＋３×ＲＥＤＣ＝８４×ＬＣ
【０１５８】
従来法２．
　（ｍ×ｋ）／ｖが２のべき乗値である必要はないので、ｖ＝１を選択する。
ステップＡ２．
　（５．５ｑ＋２．５ｖ＋１）×ＬＣ＝（２７．５＋５２．５＋１）×ＬＣ＝３１×ＬＣ
ステップＢ２．
　ｐ’－１＋Ｗ（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝（８－１＋Ｗ（（１０１０１０００）$_{2,8}$））×ＲＥＤＣ＝９×ＲＥＤＣ
合計
　３１×ＬＣ＋９×ＲＥＤＣ＝４０×ＬＣ
【０１５９】
従来法３．
　（ｍ×ｋ）／ｖが２のべき乗値である必要はないので、ｖ＝１を選択する。
ステップＡ３．
　（２．５ｋ＋２．５ｖ）×ＬＣ＝２２．５×ＬＣ
ステップＢ３及びステップＣ３
　（ｐ”＋１）×ＲＥＤＣ＝（ｌｏｇ$_2$（（ｍ×ｋ）／ｖ）＋１）×ＲＥＤＣ＝（８＋１）×ＲＥＤＣ＝９×ＲＥＤＣ
　なお（ｍ×ｋ）／ｖが２のべき乗値ではないため、補正演算を行う。
合計
　２３．５×ＬＣ＋９×ＲＥＤＣ＝３２．５×ＬＣ
【０１６０】
本発明の計算方法
ステップＡ
　（ｑ＋１）×ＬＣ＋（ｑ＋１）×ＳＣ＝６×ＬＣ＋６×ＳＣ
ステップＢ及びステップＣ
　（ｐ＋１）×ＲＥＤＣ＝（ｌｏｇ$_2$（ｍ×ｋ）＋１）×ＲＥＤＣ＝（８＋１）×ＲＥＤＣ＝９×ＲＥＤＣ
　なおｍ×ｋが２のべき乗値ではないため、補正演算を行う。
合計
　６×ＬＣ＋６×ＳＣ＋９×ＲＥＤＣ＝１５×ＬＣ＋６×ＳＣ
【実施例６】
【０１６１】
１６０ビットの楕円曲線暗号（１ワードが３２ビット：ｋ＝３２）の計算に適用
　１ワードが３２ビットの場合、１６０ビットは５ワードで示される。従ってｋ＝３２，ｍ＝５となる。また最上位から連続する「０」の個数を示すｑ＝０である。
【０１６２】
従来法１．
　ｍ×ｋ＝１６０より、（ｍ×ｋ）／ｖが２のべき乗値となるためのｖの最小値は５であるので、ｖ＝５を選択する。
ステップＡ１．

（５．５ｑ＋２．５ｖ＋１）×ＬＣ＝（１２．５＋１）×ＬＣ＝１３．５×ＬＣ
ステップＢ１
ｐ×ＲＥＤＣ＝ｌｏｇ$_2$（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝５×ＲＥＤＣ
合計
１３．５×ＬＣ＋５×ＲＥＤＣ＝１８．５×ＬＣ
【０１６３】
従来法２．
（ｍ×ｋ）／ｖが２のべき乗値である必要はないので、ｖ＝１を選択する。
ステップＡ２．
（５．５ｑ＋２．５ｖ＋１）×ＬＣ＝（２．５＋１）×ＬＣ＝３．５×ＬＣ
ステップＢ２．
ｐ’－１＋Ｗ（（ｍ×ｋ）／ｖ）×ＲＥＤＣ＝（８－１＋Ｗ（（１０１０００００）$_{2,8}$））×ＲＥＤＣ＝８×ＲＥＤＣ
合計
３．５×ＬＣ＋８×ＲＥＤＣ＝１１．５×ＬＣ
【０１６４】
従来法３．
（ｍ×ｋ）／ｖが２のべき乗値である必要はないので、ｖ＝１を選択する。
ステップＡ３．
（２．５ｋ＋２．５ｖ）×ＬＣ＝８２．５×ＬＣ
ステップＢ３及びステップＣ３
（ｐ”＋１）×ＲＥＤＣ＝（ｌｏｇ$_2$（（ｍ×ｋ）／ｖ）＋１）×ＲＥＤＣ＝（８＋１）×ＲＥＤＣ＝９×ＲＥＤＣ
なお（ｍ×ｋ）／ｖが２のべき乗値ではないため、補正演算を行う。
合計
８２．５×ＬＣ＋９×ＲＥＤＣ＝９１．５×ＬＣ
【０１６５】
本発明の計算方法
ステップＡ
（ｑ＋１）×ＬＣ＋（ｑ＋１）×ＳＣ＝ＬＣ＋ＳＣ
ステップＢ及びステップＣ
（ｐ＋１）×ＲＥＤＣ＝（ｌｏｇ$_2$（ｍ×ｋ）＋１）×ＲＥＤＣ＝（８＋１）×ＲＥＤＣ＝９×ＲＥＤＣ
なおｍ×ｋが２のべき乗値ではないため、補正演算を行う。
合計
ＬＣ＋ＳＣ＋９×ＲＥＤＣ＝１０×ＬＣ＋ＳＣ
【０１６６】
図１２は、本発明の計算方法及び従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。図１２は、実施例４乃至実施例６として示した結果を纏めて示した図表である。図１２から明らかな様に、本発明の計算方法は、実施例４乃至実施例６として示したいずれの条件下でも、従来法１乃至従来法３より優れている。
【０１６７】
前記実施の形態では、計算装置を演算カードに適用した形態を示したが、本発明はこれに限らず、パーソナルコンピュータ、サーバコンピュータ等のコンピュータ本体に適用する形態であっても良い等、様々な形態に適用することが可能である。
【０１６８】
また前記実施の形態では、ＲＥＤＣ演算を実行するコプロセッサを実装する形態を示したが、本発明はこれに限らず、ソフトウェアの処理によりＲＥＤＣ演算を実行する様にしても良い等、様々な形態に適用することが可能である。
【０１６９】

以上の実施の形態１及び２を含む実施の形態に関し、更に以下の付記を開示する。

【０１７０】

（付記１）

モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを用いて計算する計算方法において、

2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納するステップと、

レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納するステップと、

レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算するステップと

を含むことを特徴とする計算方法。

【０１７１】

（付記２）

モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を、１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタと、値Ａ及びＢ、並びに有効ワード長がｍである剰余の法ｎに対し、$2^{-m*k}\times A\times B$（ｍｏｄ　ｎ）として定義されるモンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ を実行する演算手段とを用いて計算する計算方法において、

剰余の法ｎの負数をレジスタに格納するステップと、

レジスタに格納されている値を桁上がり方向へ１ビットシフトするシフト処理を、レジスタからあふれる最上位ビットが０になるまで繰り返すステップと、

前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ＲＥＧ）ｎ を実行して、その結果をレジスタに格納する処理を、$2^{p-1}<m\times k\leqq 2^{p}$ を満たす整数であるｐ回繰り返すステップと、

$2^{p}>m\times k$ である場合に、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ｇ）ｎ を実行して、その結果をレジスタに格納するステップと（ただし$g=2^{k*G(p,m,k)}$、かつＧ（ｐ，ｍ，ｋ）＝２×ｍ－2^{p}／ｋ）、

レジスタに格納されている値を、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値として出力するステップと

を含むことを特徴とする計算方法。

【０１７２】

（付記３）

計算した同値を用いて、べき乗剰余処理を実行することを特徴とする付記１に記載の計算方法。

【０１７３】

（付記４）

モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算する計算装置において、

レジスタと、

剰余の法ｎの負数を前記レジスタに格納する手段と、

レジスタに格納されている値を桁上がり方向へ１ビットシフトする処理を、レジスタからあふれる最上位ビットが０になるまで繰り返す手段と、

レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算する手段と

を備えることを特徴とする計算装置。

【０１７４】

（付記５）

モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算する計算装置において、
１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタと、
値Ａ及びＢ、並びに有効ワード長がｍである剰余の法ｎに対し、$2^{-m*k}\times A\times B$（ｍｏｄ　ｎ）として定義されるモンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ を実行する演算手段と、
剰余の法ｎの負数をレジスタに格納する手段と、
レジスタに格納されている値を桁上がり方向へ１ビットシフトするシフト処理を、レジスタからあふれる最上位ビットが０になるまで繰り返す手段と、
前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ＲＥＧ）ｎ を実行して、その結果をレジスタに格納する処理を、$2^{p-1}<m\times k\leqq 2^{p}$ を満たす整数であるｐ回繰り返す手段と、
$2^{p}>m\times k$である場合に、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ｇ）ｎ を実行して、その結果をレジスタに格納する手段と（ただし$g=2^{k*G(p,m,k)}$、かつＧ（ｐ，ｍ，ｋ）＝２×ｍ－2^{p}／ｋ）、
レジスタに格納されている値を、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値として出力する手段と
を備えることを特徴とする計算装置。

【０１７５】

（付記６）
複数のレジスタと、
ｍ個のワードを有する第１のレジスタ及びｍ個以上のワードを有する第２のレジスタに、夫々ｎ及び０を格納する手段と、
第２のレジスタに格納されている値から、第１のレジスタに格納されている値を減じて、剰余の法ｎの負数を計算する手段と
を更に備えることを特徴とする付記５に記載の計算装置。

【０１７６】

（付記７）
レジスタに剰余の法ｎを格納する手段と、
レジスタに格納されている値の補数を、剰余の法ｎの負数として計算する手段と
を更に備えることを特徴とする付記５に記載の計算装置。

【０１７７】

（付記８）
レジスタに剰余の法ｎを格納する手段と、
レジスタに格納されている値を反転させる手段と、
レジスタに格納されている値の最下位ビットを１として、剰余の法ｎの負数を計算する手段と
を更に備えることを特徴とする付記５に記載の計算装置。

【０１７８】

（付記９）
前記シフト処理は、レジスタに格納されている値に該値を加算する加算処理であり、
前記シフト処理によりレジスタからあふれる最上位ビットは、前記加算処理により発生したキャリー値として検出する
ことを特徴とする付記５乃至付記８のいずれかに記載の計算装置。

【０１７９】

（付記１０）
１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタを備えるコンピュータに、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算させるコンピュータプログラムにおいて、

コンピュータに、2^{m*k} の法ｎに関する同値として、ｎの負数を求めてレジスタに格納させる手順と、
コンピュータに、レジスタに格納されている値を桁上がり方向へ１ビットシフトして、レジスタからあふれる最上位ビットを破棄する処理を、破棄する最上位ビットが０になるまで繰り返すことで、2^{m*k+1} の法ｎに関する同値を求めてレジスタに格納させる手順と、
コンピュータに、レジスタに格納されている値に基づくモンゴメリ乗算剰余演算により、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値を計算させる手順と
を実行させることを特徴とするコンピュータプログラム。

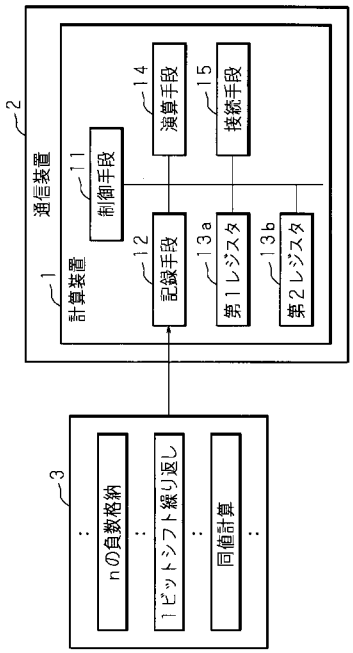
【０１８０】
（付記１１）
１ワードあたりのビット長がｋで、少なくともｍ個のワードを有するレジスタと、値Ａ及びＢ、並びに有効ワード長がｍである剰余の法ｎに対し、$2^{-m*k}\times A\times B$（ｍｏｄ　ｎ）として定義されるモンゴメリ乗算剰余演算ＲＥＤＣ（Ａ，Ｂ）ｎ を実行する演算手段とを備えるコンピュータに、モンゴメリ乗算剰余演算にて用いられ、剰余の法ｎに関する剰余値であるモンゴメリ変換パラメータに関する値を計算させるコンピュータプログラムにおいて、
コンピュータに、剰余の法ｎの負数をレジスタに格納させる手順と、
コンピュータに、レジスタに格納されている値を桁上がり方向へ１ビットシフトするシフト処理を、レジスタからあふれる最上位ビットが０になるまで繰り返させる手順と、
コンピュータに、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ＲＥＧ）ｎ を実行して、その結果をレジスタに格納する処理を、$2^{p-1}<m\times k\leqq 2^{p}$ を満たす整数であるｐ回繰り返させる手順と、
コンピュータに、$2^{p}>m\times k$ である場合に、前記演算手段により、レジスタに格納されている値ＲＥＧに対し、モンゴメリ乗算剰余演算ＲＥＤＣ（ＲＥＧ，ｇ）ｎ を実行して、その結果をレジスタに格納させる手順と（ただし$g=2^{k*G(p,m,k)}$、かつＧ（ｐ，ｍ，ｋ）＝$2\times m-2^{p}/k$）、
コンピュータに、レジスタに格納されている値を、モンゴメリ変換パラメータと、法ｎに関する剰余値が同じである同値として出力させる手順と
を実行させることを特徴とするコンピュータプログラム。

【図面の簡単な説明】
【０１８１】
【図１】本発明の計算装置の構成例を示すブロック図である。
【図２】本発明の計算方法に関するモンゴメリ乗算剰余演算のアルゴリズムを示す説明図である。
【図３】本発明の計算方法に関するモンゴメリ乗算剰余演算を用いたべき乗剰余演算のアルゴリズムを示す説明図である。
【図４】本発明の計算装置の処理を示すフローチャートである。
【図５】本発明の計算装置が備える第１レジスタ及び第２レジスタに格納される値を概念的に示す説明図である。
【図６】本発明の計算装置が備える第１レジスタ及び第２レジスタに格納される値を概念的に示す説明図である。
【図７】本発明の計算装置が備える第２レジスタに格納される値を概念的に示す説明図である。
【図８】本発明の計算装置が備える第２レジスタに格納される値を概念的に示す説明図である。
【図９】本発明の計算装置が備える第２レジスタに格納される値を概念的に示す説明図である。
【図１０】本発明のモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表で

ある。
【図１１】本発明の計算方法及び従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。
【図１２】本発明の計算方法及び従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。
【図１３】モンゴメリ乗算剰余演算のアルゴリズムを示す説明図である。
【図１４】モンゴメリ乗算剰余演算を用いたべき乗剰余演算のアルゴリズムを示す説明図である。
【図１５】モンゴメリ変換パラメータの計算方法のアルゴリズムを示す説明図である。
【図１６】従来法１におけるモンゴメリ変換パラメータの計算方法を示すフローチャートである。
【図１７】従来法１におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。
【図１８】従来法２におけるモンゴメリ変換パラメータの計算方法を示すフローチャートである。
【図１９】従来法２におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。
【図２０】従来法３におけるモンゴメリ変換パラメータの計算方法を示すフローチャートである。
【図２１】従来法３におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。
【図２２】従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表である。
【符号の説明】
【０１８２】
１　計算装置
１１　制御手段
１２　記録手段
１３ａ　第１レジスタ
１３ｂ　第２レジスタ
１４　演算手段
１５　接続手段
２　通信措置
３　コンピュータプログラム
２００　コンピュータプログラム

【図１】

本発明の計算装置の構成例を示すブロック図



【図２】

本発明の計算方法に関するモンゴメリ乗算剰余演算の
アルゴリズムを示す説明図

$input: a = (a_{m-1}, \cdots, a_1, a_0), b = (b_{m-1}, \cdots, b_1, b_0), n = (n_{m-1}, \cdots, n_1, n_0),$
$nd_0 = -n^{-1} (\mathrm{mod}\, 2^k)$
ただし、$0 < a \times b < 2^{m \times k} \times n$
$output: y = (y_m, y_{m-1}, \cdots, y_1, y_0),$

$(y_{m-1}, \cdots, y_1, y_0)_w = (0, \cdots, 0, 0)$
for $j := 0$ *to* $m-1$
　$(t2, t1) := y_0 + a_0 \times b_j$
　$u := t1 \times nd_0 (\mathrm{mod}\, 2^k)$
　$(t4, t1) := t1 + u \times ud_0$
　$(c1, c0) := t2 + t1$
　for $i := 1$ *to* $m-1$
　　$(t3, t2, t1) := y_i + (c1, c0) + a_i \times b_j$
　　$(t1, y_i) := t1 + u \times n_i$
　　$(c1, c0) := t4 + (t3, t2)$
　next i
　$(c1, c0) := (c1, c0) + y_m$
　$y_{m-1} := c0$
　$y_m := c1$
next j
if $(y \geqq n)$*then* $y := y - n$
return y

【図３】

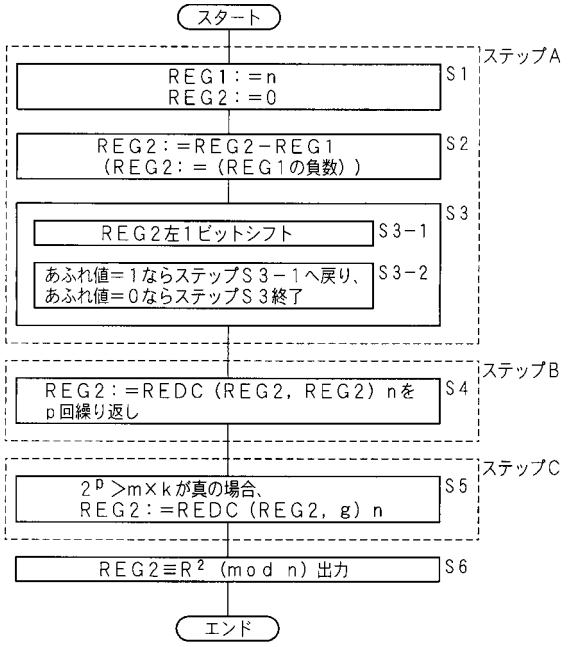
本発明の計算方法に関するモンゴメリ乗算剰余演算を
用いたべき乗剰余演算のアルゴリズムを示す説明図

$input: a, d, n$　　ただし$0 \leqq a < n$,　$d = \sum_{i=0}^{v-1} 2^i d_i (d_i = 0,1)$

$output: y = a^d (\mathrm{mod}\, n)$
1:　$y := 1$
2:　$H \equiv R^2 (\mathrm{mod}\, n)$　　モンゴメリ変換パラメータ計算
3:　$a' := REDC(a, H)_n, y' := REDC(y, H)_n$　　モンゴメリ変換
4:　*for* $i := 0$ *to* $v-1$
5:　　*if* $(d_i = 1) y' := REDC(y', a')_n$　　モンゴメリ乗算剰余
6:　　$a' := REDC(a', a')_n$　　モンゴメリ乗算剰余
7:　*next* i
8:　$y := REDC(y', 1)_n$　　モンゴメリ逆変換
9:　*return* y

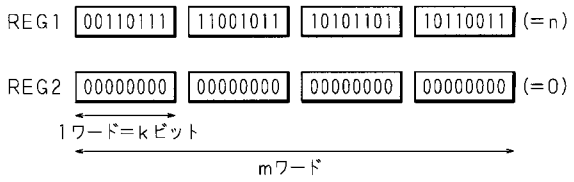
【図４】

本発明の計算装置の処理を示すフローチャート



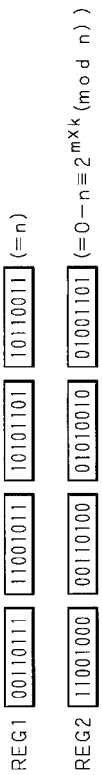
【図５】

本発明の計算装置が備える第１レジスタに及び第２レジスタに格納される値を概念的に示す説明図



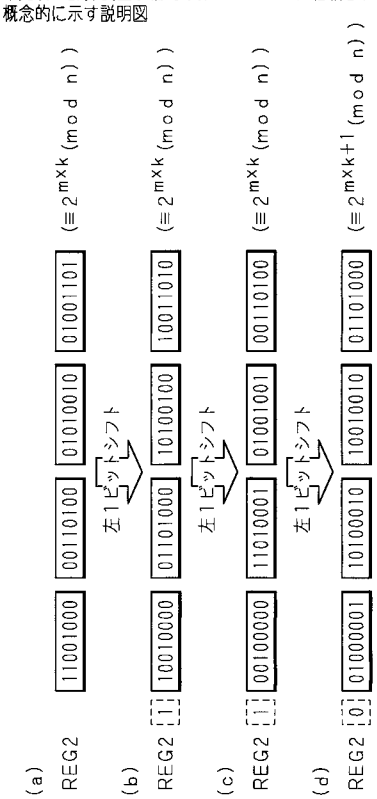
【図６】

本発明の計算装置が備える第１レジスタ及び第２レジスタに格納される値を概念的に示す説明図



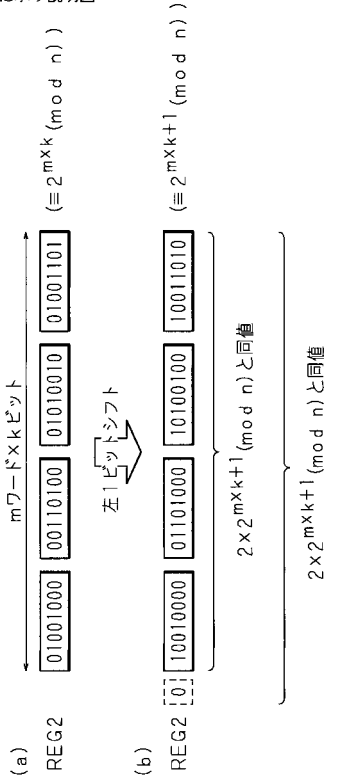
【図７】

本発明の計算装置が備える第２レジスタに格納される値を概念的に示す説明図



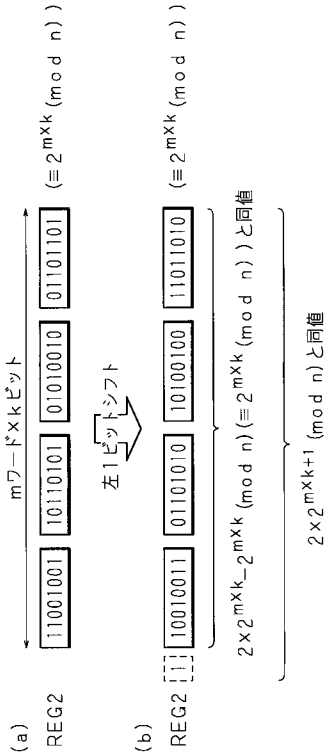
【図８】

本発明の計算装置が備える第２レジスタに格納される値を概念的に示す説明図



【図９】

本発明の計算装置が備える第２レジスタに格納される値を概念的に示す説明図



【図１０】

本発明のモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

	S1	S2	S3	S4	S5	合計
SFT	0	0	q+1	0	0	q+1
SUB(CPL)	0	1	0	0	0	1
BITCHK	0	0	q+1	0	0	q+1
REDC	0	0	0	p	$0(2^p = m \times k)$ $1(2^p = m \times k)$	$p(2^p = m \times k)$ $p+1(2^p = m \times k)$

【図１１】

本発明の計算方法及び従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

	LC及びSC （ステップA_1,A_2,A_3,A）	REDC （ステップB_1,B_2,B_3,C_3,B,C）
従来法１	$(5.5q + 2.5v + 1)\times LC$ $(q \geqq 0, v \geqq 1)$	$p(p = \log_2(m \times k)/v)$
従来法２	$(5.5q + 2.5v + 1)\times LC$ $(q \geqq 0, v \geqq 1)$	$p'^{-1}+W((m \times k)/v)$ ただし、p'は$(m \times k)/v$のビット長であり、W(x)はxをビット値で表現したときに、最上位有効ビット以外の「1」の個数を表す。
従来法３	$(2.5k+2.5v)\times LC$ $(k \geqq 8, v \geqq 1)$	$p''(2^{p''} = (m \times k)/v)$ $p''+1(2^{p''} = (m \times k)/v)$
本発明	$(q+2)\times LC+(q+1)\times SC$ $(q \geqq 0)$	$p(2^p = m \times k)$ $p+1(2^p = m \times k)$

【図１２】

本発明の計算方法及び従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

	従来法１	従来法２	従来法３	本発明
実施例４	13.5×LC	13.5×LC	92.5×LC	11×LC+SC
実施例５	84×LC	40×LC	31.5×LC	15×LC+6×SC
実施例６	18.5×LC	11.5×LC	91.5×LC	10×LC+SC

【図１３】

モンゴメリ乗算剰余演算のアルゴリズムを示す説明図

$input: a = (a_{m-1},\cdots,a_1,a_0), b = (b_{m-1},\cdots,b_1,b_0), n = (n_{m-1},\cdots,n_1,n_0),$
$nd_0 = -n^{-1} (\mathrm{mod} 2^k)$
$output: y = (y_m, y_{m-1},\cdots, y_1, y_0),$

$(y_{m-1},\cdots,y_1,y_0)_w = (0,\cdots,0,0)$
for $j := 0$ *to* $m-1$
　$(t2,t1) := y_0 + a_0 \times b_j$
　$u := t1 \times nd_0 (\mathrm{mod} 2^k)$
　$(t4,t1) := t1 + u \times ud_0$
　$(c1,c0) := t2 + t1$
　for $i := 1$ *to* $m-1$
　　$(t3,t2,t1) := y_i + (c1,c0) + a_i \times b_j$
　　$(t1,y_i) := t1 + u \times n_i$
　　$(c1,c0) := t4 + (t3,t2)$
　next i
　$(c1,c0) := (c1,c0) + y_m$
　$y_{m-1} := c0$
　$y_m := c1$
next j
if $(y \geqq n)$*then* $y := y - n$
return y

【図１４】

モンゴメリ乗算剰余演算を用いたべき乗剰余演算のアルゴリズムを示す説明図

$input: a, d, n$　　ただし$0 \leqq a < n$,　$d = \sum_{i=0}^{v-1} 2^i d_i (d_i = 0,1)$

$output: y = a^d (\mathrm{mod} n)$

1: $y := 1$
2: $H = R^2 (\mathrm{mod} n)$　　モンゴメリ変換パラメータ計算
3: $a' := REDC(a,H)_n, y' := REDC(y,H)_n$　　モンゴメリ変換
4: *for* $i := 0$ *to* $v-1$
5:　　*if* $(d_i = 1) y' := REDC(y',a')_n$　　モンゴメリ乗算剰余
6:　　$a' := REDC(a',a')_n$　　モンゴメリ乗算剰余
7: *next* i
8: $y := REDC(y',1)_n$　　モンゴメリ逆変換
9: *return* y

【図１５】

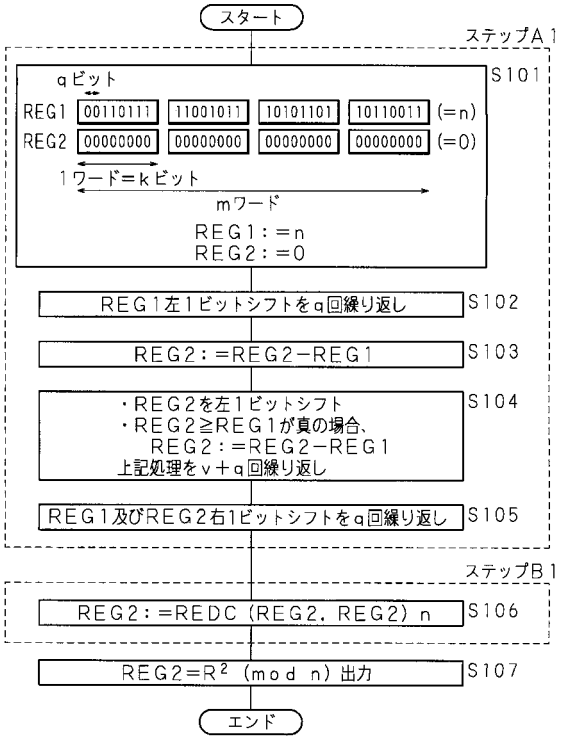
モンゴメリ変換パラメータの計算方法のアルゴリズムを示す説明図

$input: x, n$
$output: H = R^2 (\mathrm{mod} n)$、　ただし$R = 2^x (\mathrm{mod} n)$

1: $H = R (\mathrm{mod} n)$
2: *for* $i := 0$ *to* $x-1$
3: $H := H + H$
4: *if* $(H \geqq n)$*then* $H := H - n$
5: *next* i
6: *return* H

【図１６】

従来法１におけるモンゴメリ変換パラメータの計算方法を示すフローチャート



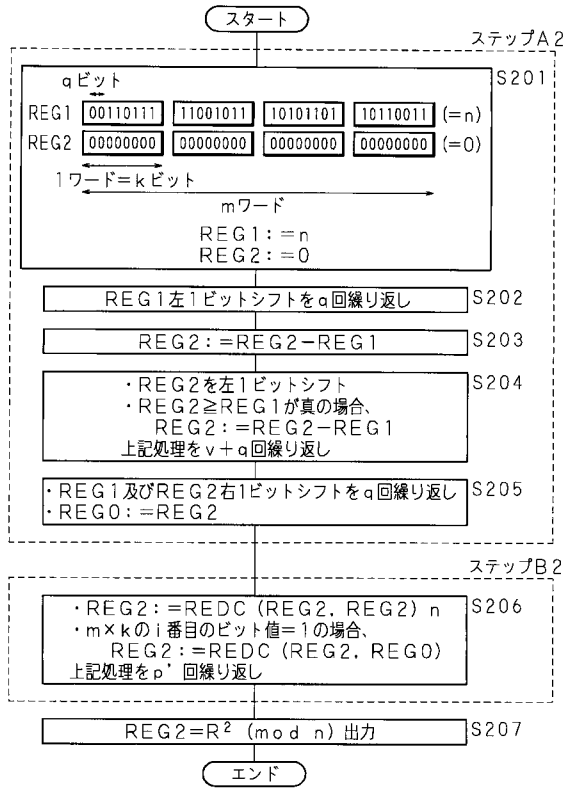
【図１７】

従来法１におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

	S101	S102	S103	S104	S105	S106	合計
SFT	0	q	0	q+v	2q	0	4q + v
SUB	0	0	1	0.5(q+v)	0	0	0.5(q+v) + 1
CMP	0	0	0	q+v	0	0	q+v
REDC	0	0	0	0	0	p	p

【図１８】

従来法２におけるモンゴメリ変換パラメータの計算方法を示すフローチャート



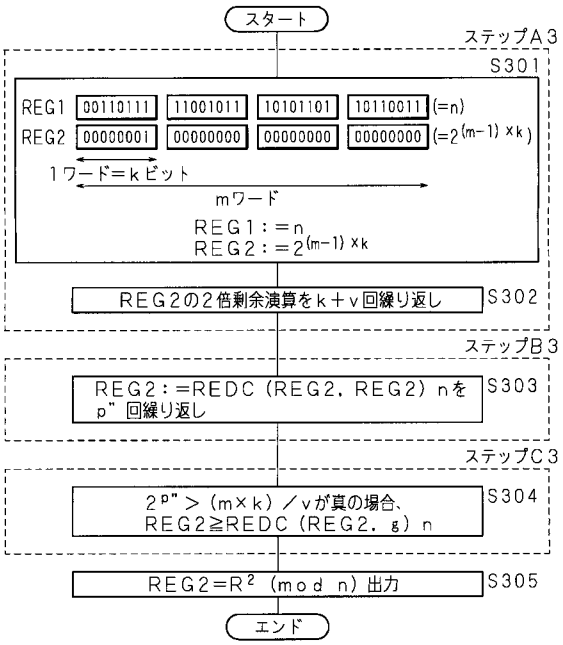
【図１９】

従来法２におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

	S201	S202	S203	S204	S205	S206	合計
SFT	0	q	0	q+v	2q	0	4q + v
SUB	0	0	1	0.5(q+v)	0	0	0.5(q+v)+ 1
CMP	0	0	0	q+v	0	0	q+v
REDC	0	0	0	0	0	p'−1+W((m×k)/v) (p'−1〜2(p' − 1))	p'−1+W((m×k)/v) (p'−1〜2(p' − 1))

【図２０】

従来法３におけるモンゴメリ変換パラメータの計算方法を示すフローチャート



【図２１】

従来法３におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

	S301	S302	S303	S304	合計
SFT	0	k+v	0	0	k+v
SUB	0	0.5(k+v)	0	0	0.5(k+v)
CMP	0	k+v	0	0	k+v
REDC	0	0	p″	$0(2^{p''}=(m\times k)/v)$ $1(2^{p''}=(m\times k)/v)$	$p''(2^{p''}=(m\times k)/v)$ $p''+1(2^{p''}=(m\times k)/v)$

【図２２】

従来法におけるモンゴメリ変換パラメータの計算方法に要する演算回数を示す図表

方法	ステップ	計算量
従来法１	ステップA1	(5.5×q＋2.5×v＋1)×LC
従来法２	ステップA2	(5.5×q＋2.5×v＋1)×LC
従来法３	ステップA3	(2.5×k＋2.5×v)×LC

フロントページの続き

(56)参考文献 特開平１０－０２１０５７（ＪＰ，Ａ）
向田健二，武仲正彦，桝井昇一，鳥居直哉，“高速モンゴメリ積和剰余演算回路の設計”，２００４年暗号と情報セキュリティシンポジウム（SCIS2004）予稿集CD-ROM，日本，２００４年 １月２７日，２Ａ３ 実装技術，２Ａ３－２
森下巖，“マイクロコンピュータの基礎”，日本，株式会社昭晃堂，１９９７年 ４月２０日，初版１１刷，ｐ．８７－９９
青木由直，恩田邦夫，“マイクロコンピュータ講義”，日本，株式会社昭晃堂，１９９６年 ２月２０日，初版１４刷，ｐ．５－７，１３６－１３７

(58)調査した分野(Int.Cl.，ＤＢ名)
Ｇ０９Ｃ １／００
Ｇ０６Ｆ ７／７２