(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号

特許第6069852号
(P6069852)

(45)発行日　平成29年2月1日(2017.2.1)　(24)登録日　平成29年1月13日(2017.1.13)

(51)Int.Cl.			F I		
H04L	9/32	(2006.01)	H04L	9/00	675B
G09C	1/00	(2006.01)	G09C	1/00	620Z

請求項の数 12　(全 61 頁)

(21)出願番号　特願2012-46687(P2012-46687)
(22)出願日　平成24年3月2日(2012.3.2)
(65)公開番号　特開2013-66151(P2013-66151A)
(43)公開日　平成25年4月11日(2013.4.11)
審査請求日　平成27年2月23日(2015.2.23)
(31)優先権主張番号　特願2011-185947(P2011-185947)
(32)優先日　平成23年8月29日(2011.8.29)
(33)優先権主張国　日本国(JP)

前置審査

(73)特許権者　000002185
ソニー株式会社
東京都港区港南１丁目７番１号
(74)代理人　100095957
弁理士　亀谷　美明
(72)発明者　作本　紘一
東京都港区港南１丁目７番１号　ソニー株式会社内
(72)発明者　白井　太三
東京都港区港南１丁目７番１号　ソニー株式会社内
(72)発明者　神尾　一也
東京都港区港南１丁目７番１号　ソニー株式会社内

最終頁に続く

(54)【発明の名称】情報処理装置、情報処理方法、及びプログラム

(57)【特許請求の範囲】
【請求項１】
多次多変数多項式の組Ｆ＝（ｆ１，…，ｆｍ）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成部と、
前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、
を備え、
前記割当部は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、複数の、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、
情報処理装置。
【請求項２】
多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成部と、
前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、

を備え、
前記割当部は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り当て処理をグループ単位で実行する、
情報処理装置。
【請求項３】
前記数生成部は、前記割当部が１つのグループに対する割り当て処理を実行する際に、当該１つのグループに属する係数の分だけ前記数を生成する、
請求項１または２に記載の情報処理装置。
【請求項４】
前記数生成部は、前記１つのグループに属する係数の分だけ前記数を生成する際、１つの数を生成し、生成した数を１ビットずつローテーションして所要数の数を生成する、
請求項３に記載の情報処理装置。
【請求項５】
前記各グループに対応する種類の項に前記係数を割り当て、当該項の変数に任意の数を代入した値をテーブルとして保持するテーブル保持部をさらに備える、
請求項１または２に記載の情報処理装置。
【請求項６】
前記情報は、乱数のシードであり、
前記所定の関数は、前記シードを利用して乱数を生成する乱数生成器である、
請求項１～５のいずれかに記載の情報処理装置。
【請求項７】
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）、及びベクトル$s \in K^n$に基づいてメッセージを生成するメッセージ生成部と、
前記多次多変数多項式の組Ｆ及びベクトル$y=(y_1,\cdots,y_m)=(f_1(s),\cdots,f_m(s))$を保持する検証者に前記メッセージを提供するメッセージ提供部と、
ｋ通り（$k \geqq 3$）の検証パターンの中から前記検証者が選択した検証パターンに対応する回答情報を前記検証者に提供する回答提供部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記回答情報は、乱数の組及び前記メッセージの中から前記検証パターンに応じて選択される情報であり、
前記メッセージは、前記公開鍵及び前記回答情報を利用して、当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
請求項１～６のいずれかに記載の情報処理装置。
【請求項８】
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）及びベクトル$y=(y_1,\cdots,y_m)=(f_1(s),\cdots,f_m(s))$を保持する情報保持部と、
前記多次多変数多項式の組Ｆ及びベクトル$s \in K^n$に基づいて生成されたメッセージを取得するメッセージ取得部と、
前記メッセージを提供した証明者に対し、ｋ通り（$k \geqq 3$）の検証パターンの中からランダムに選択した１つの検証パターンの情報を提供するパターン情報提供部と、
前記選択した検証パターンに対応する回答情報を前記証明者から取得する回答取得部と、
前記メッセージ、前記多次多変数多項式の組Ｆ、前記ベクトルｙ、及び前記回答情報に基づいて前記証明者が前記ベクトルｓを保持しているか否かを検証する検証部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、

前記メッセージは、前記公開鍵及び前記回答情報を利用して、当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
請求項１～６のいずれかに記載の情報処理装置。
【請求項９】
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）、及びベクトルｓ∈K^nに基づいてメッセージを生成するメッセージ生成部と、
前記多次多変数多項式の組Ｆ及びベクトルｙ＝（y_1，…，y_m）＝（f_1（ｓ），…，f_m（ｓ））を保持する検証者に前記メッセージを提供するメッセージ提供部と、
前記検証者がランダムに選択した第１の情報及び前記メッセージを生成する際に得られる第２の情報を用いて第３の情報を生成する中間情報生成部と、
前記検証者に前記第３の情報を提供する中間情報提供部と、
ｋ通り（ｋ≧２）の検証パターンの中から前記検証者が選択した検証パターンに対応する回答情報を前記検証者に提供する回答提供部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記回答情報は、前記メッセージの中から前記検証パターンに応じて選択される情報であり、
前記メッセージは、前記公開鍵、前記第１の情報、前記第３の情報、前記回答情報を利用して、当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
請求項１～６のいずれかに記載の情報処理装置。
【請求項１０】
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）及びベクトルｙ＝（y_1，…，y_m）＝（f_1（ｓ），…，f_m（ｓ））を保持する情報保持部と、
前記多次多変数多項式の組Ｆ及びベクトルｓ∈K^nに基づいて生成されたメッセージを取得するメッセージ取得部と、
前記メッセージを提供した証明者に対し、ランダムに選択した第１の情報を提供する情報提供部と、
前記第１の情報及び前記メッセージを生成する際に得られる第２の情報を用いて前記証明者が生成した第３の情報を取得する中間情報取得部と、
ｋ通り（ｋ≧２）の検証パターンの中からランダムに選択した１つの検証パターンの情報を前記証明者に提供するパターン情報提供部と、
前記選択した検証パターンに対応する回答情報を前記証明者から取得する回答取得部と、
前記メッセージ、前記第１の情報、前記第３の情報、前記多次多変数多項式の組Ｆ、及び前記回答情報に基づいて前記証明者が前記ベクトルｓを保持しているか否かを検証する検証部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記メッセージは、前記公開鍵、前記第１の情報、前記第３の情報、前記回答情報を利用して当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
請求項１～６のいずれかに記載の情報処理装置。
【請求項１１】
前記アルゴリズムが複数回繰り返し実行される場合において、前記数生成部は１回だけ前記数を生成し、前記割当部は１回だけ前記割り当て処理を実行し、
前記アルゴリズムは、前記割当部が割り当てた係数を繰り返し利用する、
請求項１または２に記載の情報処理装置。

【請求項１２】
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）、及び署名鍵$s \in K^n$を用いて文書Ｍに対する電子署名を生成する署名生成部と、
前記多次多変数多項式の組Ｆ及びベクトルｙ＝（f_1（ｓ），…，f_m（ｓ））を保持する検証者へと前記電子署名を提供する署名提供部と、
を備える、
請求項１～６のいずれかに記載の情報処理装置。
【発明の詳細な説明】
【技術分野】
【０００１】
本技術は、情報処理装置、情報処理方法、及びプログラムに関する。
【背景技術】
【０００２】
情報処理技術や通信技術の急速な発展に伴い、公文書、私文書を問わず、文書の電子化が急速に進んでいる。これに伴い、多くの個人や企業は、電子文書の安全管理に大きな関心を寄せている。こうした関心の高まりを受け、各方面で電子文書の盗聴や偽造等のタンパリング行為に対する対抗策が盛んに研究されるようになってきた。電子文書の盗聴に対しては、例えば、電子文書を暗号化することにより安全性が確保される。また、電子文書の偽造に対しては、例えば、電子署名を利用することにより安全性が確保される。但し、利用する暗号や電子署名が高いタンパリング耐性を有していなければ、十分な安全性が保証されない。
【０００３】
電子署名は、電子文書の作成者を特定するために利用される。そのため、電子署名は、電子文書の作成者しか生成できないようにすべきである。仮に、悪意ある第三者が同じ電子署名を生成できてしまうと、その第三者が電子文書の作成者に成りすますことができてしまう。つまり、悪意ある第三者により電子文書が偽造されてしまう。こうした偽造を防止するため、電子署名の安全性については様々な議論が交わされてきた。現在広く利用されている電子署名方式としては、例えば、ＲＳＡ署名方式やＤＳＡ署名方式などが知られている。
【０００４】
ＲＳＡ署名方式は、「大きな合成数に対する素因数分解の困難性（以下、素因数分解問題）」を安全性の根拠とする。また、ＤＳＡ署名方式は、「離散対数問題に対する解の導出の困難性」を安全性の根拠とする。これらの根拠は、古典的なコンピュータを利用して素因数分解問題や離散対数問題を効率的に解くアルゴリズムが存在しないことに起因する。つまり、上記の困難性は、古典的なコンピュータにおける計算量的な困難性を意味する。しかしながら、量子コンピュータを用いると、素因数分解問題や離散対数問題に対する解答が効率的に算出されてしまうと言われている。
【０００５】
現在利用されている電子署名方式や公開鍵認証方式の多くは、ＲＳＡ署名方式やＤＳＡ署名方式と同様、素因数分解問題や離散対数問題の困難性に安全性の根拠をおいている。そのため、こうした電子署名方式や公開鍵認証方式は、量子コンピュータが実用化された場合に、その安全性が確保されないことになる。そこで、素因数分解問題や離散対数問題など、量子コンピュータにより容易に解かれてしまう問題とは異なる問題に安全性の根拠をおく新たな電子署名方式及び公開鍵認証方式の実現が求められている。量子コンピュータにより容易に解くことが難しい問題としては、例えば、多変数多項式問題がある。
【０００６】
多変数多項式問題に安全性の根拠をおく電子署名方式としては、例えば、ＭＩ（Ｍａｔｓｕｍｏｔｏ－Ｉｍａｉ ｃｒｙｐｔｏｇｒａｐｈｙ）、ＨＦＥ（Ｈｉｄｄｅｎ Ｆｉｅｌｄ Ｅｑｕａｔｉｏｎ ｃｒｙｐｔｏｇｒａｐｈｙ）、ＯＶ（Ｏｉｌ－Ｖｉｎｅｇａｒ ｓｉｇｎａｔｕｒｅ ｓｃｈｅｍｅ）、ＴＴＭ（Ｔａｍｅｄ Ｔｒａｎｓｆｏｒｍａｔ

ion Method cryptography）に基づく方式が知られている。例えば、下記の非特許文献１、２には、ＨＦＥに基づく電子署名方式が開示されている。

【先行技術文献】

【非特許文献】

【０００７】

【非特許文献１】Jacques Patarin Asymmetric Cryptography with a Hidden Monomial. CRYPTO 1996, pp. 45－60.

【非特許文献２】Patarin, J., Courtois, N., and Goubin, L. QUARTZ, 128－Bit Long Digital Signatures. In Naccache, D., Ed. Topics in Cryptology － CT－RSA 2001 （San Francisco, CA, USA, April 2001), vol. 2020 of Lecture Notes in Computer Science, Springer－Verlag., pp. 282－297.

【発明の概要】

【発明が解決しようとする課題】

【０００８】

上記の通り、多変数多項式問題は、量子コンピュータを用いても解くことが困難なＮＰ困難問題と呼ばれる問題の一例である。通常、ＨＦＥなどに代表される多変数多項式問題を利用した公開鍵認証方式は、特殊なトラップドアが仕込まれた多次多変数連立方程式を利用している。例えば、x_1，…，x_nに関する多次多変数連立方程式Ｆ（x_1，…，x_n）＝ｙと線形変換Ａ及びＢが用意され、線形変換Ａ及びＢが秘密に管理される。この場合、多次多変数連立方程式Ｆ、線形変換Ａ及びＢがトラップドアとなる。

【０００９】

トラップドアＦ，Ａ，Ｂを知っているエンティティは、x_1，…，x_nに関する方程式Ｂ（Ｆ（Ａ（x_1，…，x_n））＝ｙ’を解くことができる。一方、トラップドアＦ，Ａ，Ｂを知らないエンティティは、x_1，…，x_nに関する方程式Ｂ（Ｆ（Ａ（x_1，…，x_n））＝ｙ’を解くことができない。この仕組みを利用することにより、多次多変数連立方程式の解答困難性を安全性の根拠とする公開鍵認証方式や電子署名方式が実現される。

【００１０】

上記の通り、こうした公開鍵認証方式や電子署名方式を実現するには、Ｂ（Ｆ（Ａ（x_1，…，x_n））＝ｙを満たすような特殊な多次多変数連立方程式を用意する必要がある。また、署名生成時に多次多変数連立方程式Ｆを解く必要がある。そのため、利用可能な多次多変数連立方程式Ｆは、比較的容易に解けるものに限られていた。すなわち、これまでの方式においては、比較的容易に解ける３つの関数（トラップドア）Ｂ、Ｆ、Ａを合成した形の多次多変数連立方程式Ｂ（Ｆ（Ａ（x_1，…，x_n））＝ｙしか用いることができず、十分な安全性を確保することが難しかった。

【００１１】

そこで、本件発明者は、上記の事情に鑑みて、効率的に解く手段（トラップドア）が知られていない多次多変数連立方程式を用いて高い安全性を有する効率的な公開鍵認証方式及び電子署名方式を考案した（Koichi Sakumoto, Taizo Shirai and Harunaga Hiwatari, “Public－Key Identification Schemes Based on Multivariate Quadratic Polynomials”, CRYPTO 2011, LNCS 6841, pp. 706－723, 2011.）。

【００１２】

しかし、この公開鍵認証方式及び電子署名方式で利用する多変数多項式は係数の数が多く、公開鍵として利用するには、証明者と検証者との間で効率的に係数を割り当てる方法

を決めておく必要があった。本技術は、効率的に多変数多項式の係数を代入することが可能な、新規かつ改良された情報処理装置、情報処理方法、及びプログラムの提供を意図して考案されたものである。

【課題を解決するための手段】

【００１３】

本技術のある観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成部と、前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、を備え、前記割当部は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、情報処理装置が提供される。

【００１４】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成部と、前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、を備え、前記割当部は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り当て処理をグループ単位で実行する、情報処理装置が提供される。

【００１５】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて前記多次多変数多項式の組Ｆを構成する各項の係数と同じ数だけ数を生成する数生成部と、前記証明者と前記認証者との間で予め決められた順序で前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、を備える、情報処理装置が提供される。

【００１６】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成するステップと、生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てるステップと、を含み、前記割り当てるステップでは、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、情報処理方法が提供される。

【００１７】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成するステップと、生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てるステップと、を含み、前記割り当てるステップでは、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り当て処理をグループ単位で実行する、情報処理方法が提供される。

【００１８】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含

む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて前記多次多変数多項式の組Ｆを構成する各項の係数と同じ数だけ数を生成するステップと、前記証明者と前記認証者との間で予め決められた順序で前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てるステップと、を含む、情報処理方法が提供される。

【００１９】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成機能と、前記数生成機能が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当機能と、をコンピュータに実現させるためのプログラムであり、前記割当機能は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、プログラムが提供される。

【００２０】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成機能と、前記数生成機能が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当機能と、をコンピュータに実現させるためのプログラムであり、前記割当機能は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り当て処理をグループ単位で実行する、プログラムが提供される。

【００２１】

また、本技術の別の観点によれば、多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて前記多次多変数多項式の組Ｆを構成する各項の係数と同じ数だけ数を生成する数生成機能と、前記証明者と前記認証者との間で予め決められた順序で前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当機能と、をコンピュータに実現させるためのプログラムが提供される。

【００２２】

また、本技術の別の観点によれば、上記のプログラムが記録された、コンピュータにより読み取り可能な記録媒体が提供される。

【発明の効果】

【００２３】

以上説明したように本技術によれば、効率的に多変数多項式の係数を代入することが可能になる。

【図面の簡単な説明】

【００２４】

【図１】公開鍵認証方式に係るアルゴリズムの構成について説明するための説明図である。

【図２】電子署名方式に係るアルゴリズムの構成について説明するための説明図である。

【図３】ｎパスの公開鍵認証方式に係るアルゴリズムの構成について説明するための説明図である。

【図４】３パスの公開鍵認証方式に係る効率的なアルゴリズムについて説明するための説明図である。

【図５】３パスの公開鍵認証方式に係る効率的なアルゴリズムの並列化について説明するための説明図である。
【図６】３パスの公開鍵認証方式に係る効率的なアルゴリズムについて説明するための説明図である。
【図７】５パスの公開鍵認証方式に係る効率的なアルゴリズムの構成例について説明するための説明図である。
【図８】５パスの公開鍵認証方式に係る効率的なアルゴリズムの並列化について説明するための説明図である。
【図９】５パスの公開鍵認証方式に係る効率的なアルゴリズムの並列化について説明するための説明図である。
【図１０】３パスの公開鍵認証方式に係る効率的なアルゴリズムを電子署名方式のアルゴリズムに変形する方法について説明するための説明図である。
【図１１】５パスの公開鍵認証方式に係る効率的なアルゴリズムを電子署名方式のアルゴリズムに変形する方法について説明するための説明図である。
【図１２】ハッシュ関数の構造例について説明するための説明図である。
【図１３】３パス方式に基づく電子署名方式に係る署名検証方法（通常の実装方法）について説明するための説明図である。
【図１４】３パス方式に基づく電子署名方式に係る署名検証方法（メモリ削減方法）について説明するための説明図である。
【図１５】５パス方式に基づく電子署名方式に係る署名検証方法（通常の実装方法）について説明するための説明図である。
【図１６】５パス方式に基づく電子署名方式に係る署名検証方法（メモリ削減方法）について説明するための説明図である。
【図１７】２進乱数から３進乱数を抽出する方法（抽出方法＃１）について説明するための説明図である。
【図１８】２進乱数から３進乱数を抽出する方法（抽出方法＃２）について説明するための説明図である。
【図１９】２進乱数から３進乱数を抽出する方法（抽出方法＃３）について説明するための説明図である。
【図２０】２進乱数から３進乱数を抽出する方法（抽出方法＃３）について説明するための説明図である。
【図２１】２進乱数から３進乱数を抽出する方法（抽出方法＃３）について説明するための説明図である。
【図２２】多変数多項式の係数を効率的に代入するためのデータ構造化手法（構造化手法＃１）について説明するための説明図である。
【図２３】多変数多項式の係数を効率的に代入するためのデータ構造化手法（構造化手法＃１）について説明するための説明図である。
【図２４】多変数多項式の係数を効率的に代入するためのデータ構造化手法（構造化手法＃１）について説明するための説明図である。
【図２５】多変数多項式の係数を効率的に代入するためのデータ構造化手法（構造化手法＃１）について説明するための説明図である。
【図２６】多変数多項式の係数を効率的に代入するためのデータ構造化手法（構造化手法＃１）について説明するための説明図である。
【図２７】多変数多項式の係数を効率的に代入するためのデータ構造化手法（構造化手法＃２）について説明するための説明図である。
【図２８】本技術の各実施形態に係るアルゴリズムを実行することが可能な情報処理装置のハードウェア構成例について説明するための説明図である。
【発明を実施するための形態】
【００２５】
以下に添付図面を参照しながら、本技術の好適な実施の形態について詳細に説明する。

なお、本明細書及び図面において、実質的に同一の機能構成を有する構成要素については、同一の符号を付することにより重複説明を省略する。

【００２６】

［説明の流れについて］

ここで、以下に記載する本技術の実施形態に関する説明の流れについて簡単に述べる。まず、図１を参照しながら、公開鍵認証方式のアルゴリズム構成について説明する。次いで、図２を参照しながら、電子署名方式のアルゴリズム構成について説明する。次いで、図３を参照しながら、ｎパスの公開鍵認証方式について説明する。

【００２７】

次いで、図４～図６を参照しながら、３パスの公開鍵認証方式に係るアルゴリズムの構成例について説明する。次いで、図７～図９を参照しながら、５パスの公開鍵認証方式に係るアルゴリズムの構成例について説明する。次いで、図１０及び図１１を参照しながら、３パス及び５パスの公開鍵認証方式に係る効率的なアルゴリズムを電子署名方式のアルゴリズムに変形する方法について説明する。

【００２８】

次いで、図１２～図１６を参照しながら、本実施形態に係る電子署名方式のアルゴリズムを実行する際に署名検証に要するメモリ量を削減する方法について説明する。次いで、図１７～図２１を参照しながら、２進乱数から３進乱数を効率的に抽出する方法について説明する。次いで、図２２～図２７を参照しながら、多変数多項式の係数を効率的に代入する方法について説明する。次いで、図２８を参照しながら、本技術の実施形態に係る各アルゴリズムを実現することが可能な情報処理装置のハードウェア構成例について説明する。最後に、本実施形態の技術的思想について纏め、当該技術的思想から得られる作用効果について簡単に説明する。

【００２９】

（説明項目）

１：はじめに
　１－１：公開鍵認証方式のアルゴリズム
　１－２：電子署名方式のアルゴリズム
　１－３：ｎパスの公開鍵認証方式
２：３パスの公開鍵認証方式に係るアルゴリズムの構成
　２－１：具体的なアルゴリズムの構成例
　２－２：並列化アルゴリズムの構成例
　２－３：３次多変数多項式に基づくアルゴリズムの構成例
３：５パスの公開鍵認証方式に係るアルゴリズムの構成
　３－１：具体的なアルゴリズムの構成例
　３－２：並列化アルゴリズムの構成例
　３－３：３次多変数多項式に基づくアルゴリズムの構成例
４：電子署名方式への変形
　４－１：３パスの公開鍵認証方式から電子署名方式への変形
　４－２：５パスの公開鍵認証方式から電子署名方式への変形
５：署名検証に要するメモリ量の削減方法について
　５－１：ハッシュ関数の構造について
　５－２：３パス方式に基づく電子署名方式への適用例
　５－３：５パス方式に基づく電子署名方式への適用例
６：２進数の乱数列から３進数の乱数列を抽出する方法について
　６－１：抽出方法＃１（２ビット区切り）
　６－２：抽出方法＃２（区切りなし）
　６－３：抽出方法＃３（ｋビット区切り）
　　６－３－１：基本構成
　　６－３－２：追加抽出方法

７：多変数多項式の係数を効率的に代入する方法について
７－１：基本的な取り決め
７－２：データの構造化
７－２－１：構造化手法＃１
７－２－２：構造化手法＃２
７－２－３：構造化手法＃３
８：ハードウェア構成例
９：まとめ

【００３０】

＜１：はじめに＞

本実施形態は、多次多変数連立方程式に対する求解問題の困難性に安全性の根拠をおく公開鍵認証方式及び電子署名方式に関する。但し、本実施形態は、ＨＦＥ電子署名方式などの従来手法とは異なり、効率的に解く手段（トラップドア）を持たない多次多変数連立方程式を利用する公開鍵認証方式及び電子署名方式に関する。まず、公開鍵認証方式のアルゴリズム、電子署名方式のアルゴリズム、及びｎパスの公開鍵認証方式について、その概要を簡単に説明する。

【００３１】

［１－１：公開鍵認証方式のアルゴリズム］

まず、図１を参照しながら、公開鍵認証方式のアルゴリズムについて概要を説明する。図１は、公開鍵認証方式のアルゴリズムについて概要を説明するための説明図である。

【００３２】

公開鍵認証は、ある人（証明者）が、公開鍵ｐｋ及び秘密鍵ｓｋを利用して、他の人（検証者）に本人であることを納得させるために利用される。例えば、証明者Ａの公開鍵pk_Aは、検証者Ｂに公開される。一方、証明者Ａの秘密鍵sk_Aは、証明者Ａにより秘密に管理される。公開鍵認証の仕組みにおいては、公開鍵pk_Aに対応する秘密鍵sk_Aを知る者が証明者Ａ本人であるとみなされる。

【００３３】

公開鍵認証の仕組みを利用して証明者Ａが証明者Ａ本人であることを検証者Ｂに証明するには、対話プロトコルを介して、証明者Ａが公開鍵pk_Aに対応する秘密鍵sk_Aを知っているという証拠を検証者Ｂに提示すればよい。そして、証明者Ａが秘密鍵sk_Aを知っているという証拠が検証者Ｂに提示され、その証拠を検証者Ｂが確認し終えた場合、証明者Ａの正当性（本人であること）が証明されたことになる。

【００３４】

但し、公開鍵認証の仕組みには、安全性を担保するために以下の条件が求められる。

【００３５】

１つ目の条件は、「対話プロトコルを実行した際に秘密鍵ｓｋを持たない偽証者により偽証が成立してしまう確率を限りなく小さくする」ことである。この１つ目の条件が成り立つことを「健全性」と呼ぶ。つまり、健全性とは、「秘密鍵ｓｋを持たない偽証者により、対話プロトコルの実行中に無視できない確率で偽証が成立することはないこと」と言い換えられる。２つ目の条件は、「対話プロトコルを実行したとしても、証明者Ａが有する秘密鍵sk_Aの情報が検証者Ｂに一切漏れることがない」ことである。この２つ目の条件が成り立つことを「零知識性」と呼ぶ。

【００３６】

安全に公開鍵認証を行うには、健全性及び零知識性を有する対話プロトコルを利用する必要がある。仮に、健全性及び零知識性を有しない対話プロトコルを用いて認証処理を行った場合には、偽証された可能性及び秘密鍵の情報が漏れてしまった可能性が否定できないため、処理自体が成功裡に完了しても証明者の正当性を証明したことにはならない。従って、対話プロトコルの健全性及び零知識性を如何に保証するかが重要になる。

【００３７】

（モデル）

公開鍵認証方式のモデルには、図１に示すように、証明者と検証者という２つのエンティティが存在する。証明者は、鍵生成アルゴリズムＧｅｎを用いて、証明者固有の秘密鍵ｓｋと公開鍵ｐｋの組を生成する。次いで、証明者は、鍵生成アルゴリズムＧｅｎを用いて生成した秘密鍵ｓｋと公開鍵ｐｋの組を利用して検証者と対話プロトコルを実行する。このとき、証明者は、証明者アルゴリズムＰを利用して対話プロトコルを実行する。上記の通り、証明者は、証明者アルゴリズムＰを利用し、対話プロトコルの中で秘密鍵ｓｋを保有している証拠を検証者に提示する。

【００３８】

一方、検証者は、検証者アルゴリズムＶを利用して対話プロトコルを実行し、証明者が公開している公開鍵に対応する秘密鍵を、その証明者が保有しているか否かを検証する。つまり、検証者は、証明者が公開鍵に対応する秘密鍵を保有しているか否かを検証するエンティティである。このように、公開鍵認証方式のモデルは、証明者と検証者という２つのエンティティ、及び、鍵生成アルゴリズムＧｅｎ、証明者アルゴリズムＰ、検証者アルゴリズムＶという３つのアルゴリズムにより構成される。

【００３９】

なお、以下の説明において、「証明者」「検証者」という表現を用いるが、これらの表現はあくまでもエンティティを意味するものである。従って、鍵生成アルゴリズムＧｅｎ、証明者アルゴリズムＰを実行する主体は、「証明者」のエンティティに対応する情報処理装置である。同様に、検証者アルゴリズムＶを実行する主体は、情報処理装置である。これら情報処理装置のハードウェア構成は、例えば、図２８に示した通りである。つまり、鍵生成アルゴリズムＧｅｎ、証明者アルゴリズムＰ、検証者アルゴリズムＶは、ＲＯＭ９０４、ＲＡＭ９０６、記憶部９２０、リムーバブル記録媒体９２８などに記録されたプログラムに基づいてＣＰＵ９０２などにより実行される。

【００４０】

（鍵生成アルゴリズムＧｅｎ）

鍵生成アルゴリズムＧｅｎは、証明者により利用される。鍵生成アルゴリズムＧｅｎは、証明者に固有の秘密鍵ｓｋと公開鍵ｐｋとの組を生成するアルゴリズムである。鍵生成アルゴリズムＧｅｎにより生成された公開鍵ｐｋは公開される。そして、公開された公開鍵ｐｋは、検証者により利用される。一方、鍵生成アルゴリズムＧｅｎにより生成された秘密鍵ｓｋは、証明者が秘密に管理する。そして、証明者により秘密に管理される秘密鍵ｓｋは、公開鍵ｐｋに対応する秘密鍵ｓｋを証明者が保有していることを検証者に対して証明するために利用される。形式的に、鍵生成アルゴリズムＧｅｎは、セキュリティパラメータ1^{λ}（λは０以上の整数）を入力とし、秘密鍵ｓｋと公開鍵ｐｋを出力するアルゴリズムとして、下記の式（１）のように表現される。

【００４１】

【数１】

$$(sk, pk) \leftarrow Gen(1^{\lambda}) \quad \cdots (1)$$

【００４２】

（証明者アルゴリズムＰ）

証明者アルゴリズムＰは、証明者により利用される。証明者アルゴリズムＰは、公開鍵ｐｋに対応する秘密鍵ｓｋを証明者が保有していることを検証者に対して証明するためのアルゴリズムである。つまり、証明者アルゴリズムＰは、秘密鍵ｓｋと公開鍵ｐｋとを入力とし、対話プロトコルを実行するアルゴリズムである。

【００４３】

（検証者アルゴリズムＶ）

検証者アルゴリズムＶは、検証者により利用される。検証者アルゴリズムＶは、対話プロトコルの中で、公開鍵ｐｋに対応する秘密鍵ｓｋを証明者が保有しているか否かを検証

するアルゴリズムである。検証者アルゴリズムＶは、公開鍵ｐｋを入力とし、対話プロトコルの実行結果に応じて０又は１（１ｂｉｔ）を出力するアルゴリズムである。なお、検証者は、検証者アルゴリズムＶが０を出力した場合には証明者が不正なものであると判断し、１を出力した場合には証明者が正当なものであると判断する。形式的に、検証者アルゴリズムＶは、下記の式（２）のように表現される。

【００４４】

【数２】

$$0/1 \leftarrow V(pk) \quad \cdots (2)$$

【００４５】

上記の通り、意味のある公開鍵認証を実現するには、対話プロトコルが健全性及び零知識性という２つの条件を満たしている必要がある。しかし、証明者が秘密鍵ｓｋを保有していることを証明するためには、証明者が秘密鍵ｓｋに依存した手続きを実行し、その結果を検証者に通知した上で、その通知内容に基づく検証を検証者に実行させる必要がある。秘密鍵ｓｋに依存した手続きを実行するのは、健全性を担保するために必要である。一方で、秘密鍵ｓｋの情報が一切検証者に漏れないようにする必要がある。そのため、これらの要件を満たすように、上記の鍵生成アルゴリズムＧｅｎ、証明者アルゴリズムＰ、検証者アルゴリズムＶを巧妙に設計する必要がある。

【００４６】

以上、公開鍵認証方式のアルゴリズムについて、その概要を説明した。

【００４７】

［１－２：電子署名方式のアルゴリズム］

次に、図２を参照しながら、電子署名方式のアルゴリズムについて概要を説明する。図２は、電子署名方式のアルゴリズムについて概要を説明するための説明図である。

【００４８】

紙文書とは異なり、ある電子化されたデータに対して押印したり署名を記載したりすることはできない。そのため、電子化されたデータの作成者を証明するためには、紙文書に押印したり署名を記載したりするのと同等の効果が得られる電子的な仕組みを必要とする。この仕組みが電子署名である。電子署名とは、データの作成者しか知らない署名データをデータに関連付けて受領者に提供し、その署名データを受領者側で検証する仕組みのことを言う。

【００４９】

（モデル）

電子署名方式のモデルには、図２に示すように、署名者及び検証者という２つのエンティティが存在する。そして、電子署名方式のモデルは、鍵生成アルゴリズムＧｅｎ、署名生成アルゴリズムＳｉｇ、署名検証アルゴリズムＶｅｒという３つのアルゴリズムにより構成される。

【００５０】

署名者は、鍵生成アルゴリズムＧｅｎを利用して署名者固有の署名鍵ｓｋと検証鍵ｐｋとの組を生成する。また、署名者は、署名生成アルゴリズムＳｉｇを利用して文書Ｍに付与する電子署名σを生成する。つまり、署名者は、文書Ｍに電子署名を付与するエンティティである。一方、検証者は、署名検証アルゴリズムＶｅｒを利用して文書Ｍに付与された電子署名σを検証する。つまり、検証者は、文書Ｍの作成者が署名者であるか否かを確認するために、電子署名σを検証するエンティティである。

【００５１】

なお、以下の説明において、「署名者」「検証者」という表現を用いるが、これらの表現はあくまでもエンティティを意味するものである。従って、鍵生成アルゴリズムＧｅｎ、署名生成アルゴリズムＳｉｇを実行する主体は、「署名者」のエンティティに対応する

情報処理装置である。同様に、署名検証アルゴリズムＶｅｒを実行する主体は、情報処理装置である。これら情報処理装置のハードウェア構成は、例えば、図２８に示した通りである。つまり、鍵生成アルゴリズムＧｅｎ、署名生成アルゴリズムＳｉｇ、署名検証アルゴリズムＶｅｒは、ＲＯＭ９０４、ＲＡＭ９０６、記憶部９２０、リムーバブル記録媒体９２８などに記録されたプログラムに基づいてＣＰＵ９０２などにより実行される。

【００５２】

（鍵生成アルゴリズムＧｅｎ）

鍵生成アルゴリズムＧｅｎは、署名者により利用される。鍵生成アルゴリズムＧｅｎは、署名者固有の署名鍵ｓｋと検証鍵ｐｋとの組を生成するアルゴリズムである。鍵生成アルゴリズムＧｅｎにより生成された検証鍵ｐｋは公開される。一方、鍵生成アルゴリズムＧｅｎにより生成された署名鍵ｓｋは、署名者により秘密に管理される。そして、署名鍵ｓｋは、文書Ｍに付与される電子署名σの生成に利用される。例えば、鍵生成アルゴリズムＧｅｎは、セキュリティパラメータ1^{λ}（λは０以上の整数）を入力とし、署名鍵ｓｋ及び公開鍵ｐｋを出力する。この場合、鍵生成アルゴリズムＧｅｎは、形式的に、下記の式（３）のように表現することができる。

【００５３】

【数３】

$$(sk, pk) \leftarrow Gen(1^{\lambda})$$

…（３）

【００５４】

（署名生成アルゴリズムＳｉｇ）

署名生成アルゴリズムＳｉｇは、署名者により利用される。署名生成アルゴリズムＳｉｇは、文書Ｍに付与される電子署名σを生成するアルゴリズムである。署名生成アルゴリズムＳｉｇは、署名鍵ｓｋと文書Ｍとを入力とし、電子署名σを出力するアルゴリズムである。この署名生成アルゴリズムＳｉｇは、形式的に、下記の式（４）のように表現することができる。

【００５５】

【数４】

$$\sigma \leftarrow Sig(sk, M)$$

…（４）

【００５６】

（署名検証アルゴリズムＶｅｒ）

署名検証アルゴリズムＶｅｒは、検証者により利用される。署名検証アルゴリズムＶｅｒは、電子署名σが文書Ｍに対する正当な電子署名であるか否かを検証するアルゴリズムである。署名検証アルゴリズムＶｅｒは、署名者の検証鍵ｐｋ、文書Ｍ、電子署名σを入力とし、０又は１（１ｂｉｔ）を出力するアルゴリズムである。この署名検証アルゴリズムＶｅｒは、形式的に、下記の式（５）のように表現することができる。なお、検証者は、署名検証アルゴリズムＶｅｒが０を出力した場合（公開鍵ｐｋが文書Ｍと電子署名σを拒否する場合）に電子署名σが不当であると判断し、１を出力した場合（公開鍵ｐｋが文書Ｍと電子署名σを受理する場合）に電子署名σが正当であると判断する。

【００５７】

【数５】

$$0/1 \leftarrow Ver(pk, M, \sigma)$$

…（５）

【００５８】
以上、電子署名方式のアルゴリズムについて、その概要を説明した。
【００５９】
［１－３：ｎパスの公開鍵認証方式］
次に、図３を参照しながら、ｎパスの公開鍵認証方式について説明する。図３は、ｎパスの公開鍵認証方式について説明するための説明図である。
【００６０】
上記の通り、公開鍵認証方式は、対話プロトコルの中で、証明者が公開鍵ｐｋに対応する秘密鍵ｓｋを保有していることを検証者に証明する認証方式である。また、対話プロトコルは、健全性及び零知識性という２つの条件を満たす必要がある。そのため、対話プロトコルの中では、図３に示すように、証明者及び検証者の双方がそれぞれ処理を実行しながらｎ回の情報交換を行う。
【００６１】
ｎパスの公開鍵認証方式の場合、証明者アルゴリズムＰを用いて証明者により処理（工程＃１）が実行され、情報T_1が検証者に送信される。次いで、検証者アルゴリズムＶを用いて検証者により処理（工程＃２）が実行され、情報T_2が証明者に送信される。さらに、ｋ＝３～ｎについて処理の実行及び情報T_kの送信が順次行われ、最後に処理（工程＃ｎ＋１）が実行される。このように、情報がｎ回送受信される方式のことを「ｎパス」の公開鍵認証方式と呼ぶ。
【００６２】
以上、ｎパスの公開鍵認証方式について説明した。
【００６３】
＜２：３パスの公開鍵認証方式に係るアルゴリズムの構成＞
以下、３パスの公開鍵認証方式に係るアルゴリズムについて説明する。なお、以下の説明において、３パスの公開鍵認証方式のことを「３パス方式」と呼ぶ場合がある。
【００６４】
［２－１：具体的なアルゴリズムの構成例（図４）］
まず、図４を参照しながら、３パス方式に係る具体的なアルゴリズムの構成例について紹介する。図４は、３パス方式に係る具体的なアルゴリズムの構成について説明するための説明図である。ここでは、公開鍵ｐｋの一部として２次多項式の組（$f_1(x)$，…，$f_m(x)$）を利用する場合について考える。但し、２次多項式$f_i(x)$は、下記の式（６）のように表現されるものとする。また、ベクトル（x_1，…，x_n）をｘと表記し、２次多項式の組（$f_1(x)$，…，$f_m(x)$）を多変数多項式Ｆ（ｘ）と表記することにする。
【００６５】
【数６】

$$f_i(x_1,\cdots,x_n)=\sum_{j,k}a_{ijk}x_jx_k+\sum_j b_{ij}x_j \quad \cdots(6)$$

【００６６】
また、２次多項式の組（$f_1(x)$，…，$f_m(x)$）は、下記の式（７）のように表現することができる。また、A_1，…，A_mは、ｎ×ｎ行列である。さらに、b_1，…，b_mはそれぞれｎ×１ベクトルである。
【００６７】

【数７】

$$F(x)=\begin{pmatrix} f_1(x) \\ \vdots \\ f_m(x) \end{pmatrix}=\begin{pmatrix} x^T A_1 x+b_1^T x \\ \vdots \\ x^T A_m x+b_m^T x \end{pmatrix}$$

…（７）

【００６８】

この表現を用いると、多変数多項式Ｆは、下記の式（８）及び式（９）のように表現することができる。この表現が成り立つことは、下記の式（１０）から容易に確認することができる。

【００６９】

【数８】

$$F(x+y)=F(x)+F(y)+G(x,y)$$

…（８）

$$G(x,y)=\begin{pmatrix} y^T(A_1^T+A_1)x \\ \vdots \\ y^T(A_m^T+A_m)x \end{pmatrix}$$

…（９）

$$\begin{aligned} f_l(x+y) &= (x+y)^T A_l(x+y)+b_l^T(x+y) \\ &= x^T A_l x+x^T A_l y+y^T A_l x+y^T A_l y+b_l^T x+b_l^T y \\ &= f_l(x)+f_l(y)+x^T A_l y+y^T A_l x \\ &= f_l(x)+f_l(y)+x^T(A_l^T)^T y+y^T A_l x \\ &= f_l(x)+f_l(y)+(A_l^T x)^T y+y^T A_l x \\ &= f_l(x)+f_l(y)+y^T(A_l^T x)+y^T A_l x \\ &= f_l(x)+f_l(y)+y^T(A_l^T+A_l)x \end{aligned}$$

…（１０）

【００７０】

このようにＦ（ｘ＋ｙ）をｘに依存する第１の部分と、ｙに依存する第２の部分と、ｘ及びｙの両方に依存する第３の部分とに分けたとき、第３の部分に対応する項Ｇ（ｘ，ｙ）は、ｘ及びｙについて双線形になる。以下、項Ｇ（ｘ，ｙ）を双線形項と呼ぶ場合がある。この性質を利用すると、効率的なアルゴリズムを構築することが可能になる。

【００７１】

例えば、ベクトル$t_0 \in K^n$、$e_0 \in K^m$を用いて、多変数多項式Ｆ（ｘ＋ｒ）のマスクに利用する多変数多項式F_1（ｘ）を$F_1(x)=G(x,t_0)+e_0$と表現する。この場合、多変数多項式Ｆ（ｘ＋r_0）とＧ（ｘ）との和は、下記の式（１１）のように表現される。ここで、$t_1=r_0+t_0$、$e_1=F(r_0)+e_0$とおけば、多変数多項式$F_2(x)=F(x+r_0)+F_1(x)$は、ベクトル$t_1 \in K^n$、$e_1 \in K^m$により表現することができる。そのため、$F_1(x)=G(x,t_0)+e_0$に設定すれば、K^n上のベクトル及びK^m上のベクトルを用いてF_1及びF_2を表現できるようになり、通信に必要なデータサイズの少ない効率的なアルゴリズムを実現することが可能になる。

【００７２】

【数９】

$$\begin{aligned} &F(x+r_0)+F_1(x) \\ &=F(x)+F(r_0)+G(x,r_0)+G(x,t_0)+e_0 \\ &=F(x)+G(x,r_0+t_0)+F(r_0)+e_0 \end{aligned}$$

…（１１）

【００７３】

なお、F_2（或いはF_1）からr_0に関する情報が一切漏れることはない。例えば、e_1及びt_1（或いはe_0及びt_0）を与えられても、e_0及びt_0（或いはe_1及びt_1）を知らない限り、r_0の情報を一切知ることはできない。従って、零知識性が担保される。以下、上記の論理に基づいて構築された３パス方式のアルゴリズムについて説明する。ここで説明する３パス方式のアルゴリズムは、以下のような鍵生成アルゴリズムＧｅｎ、証明者アルゴリズムＰ、検証者アルゴリズムＶにより構成される。

【００７４】

（鍵生成アルゴリズムＧｅｎ）

鍵生成アルゴリズムＧｅｎは、環Ｋ上で定義されるｍ本の多変数多項式$f_1(x_1,\cdots,x_n)$，…，$f_m(x_1,\cdots,x_n)$、及びベクトル$s=(s_1,\cdots,s_n) \in K^n$を生成する。次に、鍵生成アルゴリズムＧｅｎは、$y=(y_1,\cdots,y_m) \leftarrow (f_1(s),\cdots,f_m(s))$を計算する。そして、鍵生成アルゴリズムＧｅｎは、$(f_1(x_1,\cdots,x_n),\cdots,f_m(x_1,\cdots,x_n),y)$を公開鍵ｐｋに設定し、ｓを秘密鍵に設定する。

【００７５】

（証明者アルゴリズムＰ、検証者アルゴリズムＶ）

以下、図４を参照しながら、対話プロトコルの中で証明者アルゴリズムＰが実行する処理及び検証者アルゴリズムＶが実行する処理について説明する。この対話プロトコルの中で、証明者は、秘密鍵ｓの情報を検証者に一切漏らさずに、「自身がｙ＝Ｆ（ｓ）を満たすｓを知っていること」を検証者に示す。一方、検証者は、証明者がｙ＝Ｆ（ｓ）を満たすｓを知っているか否かを検証する。なお、公開鍵ｐｋは、検証者に公開されているものとする。また、秘密鍵ｓは、証明者により秘密に管理されているものとする。以下、図４に示したフローチャートに沿って説明を進める。

【００７６】

工程＃１：

図４に示すように、まず、証明者アルゴリズムＰは、ランダムにベクトルr_0，$t_0 \in$

K^n及び$e_0 \in K^m$を生成する。次いで、証明者アルゴリズムＰは、$r_1 \leftarrow s - r_0$を計算する。この計算は、秘密鍵ｓをベクトルr_0によりマスクする操作に相当する。さらに、証明者アルゴリズムＰは、$t_1 \leftarrow r_0 - t_0$を計算する。次いで、証明者アルゴリズムＰは、$e_1 \leftarrow F(r_0) - e_0$を計算する。

【００７７】

工程＃１（続き）：

次いで、証明者アルゴリズムＰは、$c_0 \leftarrow H(r_1, G(t_0, r_1) + e_0)$を計算する。次いで、証明者アルゴリズムＰは、$c_1 \leftarrow H(t_0, e_0)$を計算する。次いで、証明者アルゴリズムＰは、$c_2 \leftarrow H(t_1, e_1)$を計算する。工程＃１で生成されたメッセージ（$c_0$，$c_1$，$c_2$）は、検証者アルゴリズムＶに送られる。

【００７８】

工程＃２：

メッセージ（c_0，c_1，c_2）を受け取った検証者アルゴリズムＶは、３つの検証パターンのうち、どの検証パターンを利用するかを選択する。例えば、検証者アルゴリズムＶは、検証パターンの種類を表す３つの数値｛０，１，２｝の中から１つの数値を選択し、選択した数値を要求Ｃｈに設定する。この要求Ｃｈは証明者アルゴリズムＰに送られる。

【００７９】

工程＃３：

要求Ｃｈを受け取った証明者アルゴリズムＰは、受け取った要求Ｃｈに応じて検証者アルゴリズムＶに送る返答Ｒｓｐを生成する。Ｃｈ＝０の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（r_0，t_1，e_1）を生成する。Ｃｈ＝１の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（r_1，t_0，e_0）を生成する。Ｃｈ＝２の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（r_1，t_1，e_1）を生成する。工程＃３で生成された返答Ｒｓｐは、検証者アルゴリズムＶに送られる。

【００８０】

工程＃４：

返答Ｒｓｐを受け取った検証者アルゴリズムＶは、受け取った返答Ｒｓｐを利用して以下の検証処理を実行する。

【００８１】

Ｃｈ＝０の場合、検証者アルゴリズムＶは、$c_1 = H(r_0 - t_1, F(r_0) - e_1)$の等号が成り立つか否かを検証する。さらに、検証者アルゴリズムＶは、$c_2 = H(t_1, e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。

【００８２】

Ｃｈ＝１の場合、検証者アルゴリズムＶは、$c_0 = H(r_1, G(t_0, r_1) + e_0)$の等号が成り立つか否かを検証する。さらに、検証者アルゴリズムＶは、$c_1 = H(t_0, e_0)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。

【００８３】

Ｃｈ＝２の場合、検証者アルゴリズムＶは、$c_0 = H(r_1, y - F(r_1) - G(t_1, r_1) - e_1)$の等号が成り立つか否かを検証する。さらに、検証者アルゴリズムＶは、$c_2 = H(t_1, e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。

【００８４】

以上、３パス方式に係る効率的なアルゴリズムの構成例について説明した。

【００８５】

［２－２：並列化アルゴリズムの構成例（図５）］

次に、図５を参照しながら、図４に示した３パス方式のアルゴリズムを並列化する方法について説明する。なお、鍵生成アルゴリズムＧｅｎの構成については説明を省略する。

【００８６】

さて、上記の対話プロトコルを適用すれば、偽証が成功する確率を２／３以下に抑制することができる。従って、この対話プロトコルを２回実行すれば、偽証が成功する確率を$（２／３）^{2}$以下に抑制することができる。さらに、この対話プロトコルをＮ回実行すると、偽証が成功する確率は$（２／３）^{N}$となり、Ｎを十分に大きい数（例えば、Ｎ＝１４０）にすれば、偽証が成功する確率は無視できる程度に小さくなる。

【００８７】

対話プロトコルを複数回実行する方法としては、例えば、メッセージ、要求、返答のやり取りを逐次的に複数回繰り返す直列的な方法と、１回分のやり取りで複数回分のメッセージ、要求、返答のやり取りを行う並列的な方法とが考えられる。さらに、直列的な方法と並列的な方法とを組み合わせたハイブリッド型の方法も考えられる。ここでは、図５を参照しながら、３パス方式に係る上記の対話プロトコルを並列的に実行するアルゴリズム（以下、並列化アルゴリズム）について説明する。

【００８８】

工程＃１：

図５に示すように、まず、証明者アルゴリズムＰは、ｉ＝１～Ｎについて以下の処理（１）～処理（６）を実行する。

処理（１）：証明者アルゴリズムＰは、ランダムにベクトル$ｒ_{0\,i}$，$ｔ_{0\,i}∈Ｋ^{n}$及び$ｅ_{0\,i}∈Ｋ^{m}$を生成する。

処理（２）：証明者アルゴリズムＰは、$ｒ_{1\,i}←ｓ－ｒ_{0\,i}$を計算する。この計算は、秘密鍵ｓをベクトル$ｒ_{0\,i}$によりマスクする操作に相当する。さらに、証明者アルゴリズムＰは、$ｔ_{1\,i}←ｒ_{0\,i}＋ｔ_{0\,i}$を計算する。

処理（３）：証明者アルゴリズムＰは、$ｅ_{1\,i}←Ｆ（ｒ_{0\,i}）－ｅ_{0\,i}$を計算する。

処理（４）：証明者アルゴリズムＰは、$ｃ_{0\,i}←Ｈ（ｒ_{1\,i}，Ｇ（ｒ_{1\,i}，ｔ_{0\,i}）＋ｅ_{0\,i}）$を計算する。

処理（５）：証明者アルゴリズムＰは、$ｃ_{1\,i}←Ｈ（ｔ_{0\,i}，ｅ_{0\,i}）$を計算する。

処理（６）：証明者アルゴリズムＰは、$ｃ_{2\,i}←Ｈ（ｔ_{1\,i}，ｅ_{1\,i}）$を計算する。

【００８９】

工程＃１（続き）

ｉ＝１～Ｎについて上記の処理（１）～処理（６）を実行した後、証明者アルゴリズムＰは、$Ｃｍｔ←Ｈ（ｃ_{0\,1}，ｃ_{1\,1}，ｃ_{2\,1}，…，ｃ_{0\,N}，ｃ_{1\,N}，ｃ_{2\,N}）$を計算する。工程＃１で生成されたハッシュ値Ｃｍｔは、検証者アルゴリズムＶに送られる。このように、メッセージ$（ｃ_{0\,1}，ｃ_{1\,1}，ｃ_{2\,1}，…，ｃ_{0\,N}，ｃ_{1\,N}，ｃ_{2\,N}）$をハッシュ値に変換してから検証者アルゴリズムＶに送ることで、通信量を削減することが可能になる。

【００９０】

工程＃２：

ハッシュ値Ｃｍｔを受け取った検証者アルゴリズムＶは、ｉ＝１～Ｎのそれぞれについて、３つの検証パターンのうち、どの検証パターンを利用するかを選択する。例えば、検証者アルゴリズムＶは、ｉ＝１～Ｎのそれぞれについて、検証パターンの種類を表す３つの数値｛０，１，２｝の中から１つの数値を選択し、選択した数値を要求$Ｃｈ_{i}$に設定する。要求$Ｃｈ_{1}$，…，$Ｃｈ_{N}$は、証明者アルゴリズムＰに送られる。

【００９１】

工程＃３：

要求$Ｃｈ_{1}$，…，$Ｃｈ_{N}$を受け取った証明者アルゴリズムＰは、受け取った要求$Ｃｈ_{1}$，…，$Ｃｈ_{N}$のそれぞれ応じて検証者アルゴリズムＶに送る返答$Ｒｓｐ_{1}$，…，$Ｒｓｐ_{N}$を生成する。$Ｃｈ_{i}＝０$の場合、証明者アルゴリズムＰは、$Ｒｓｐ_{i}＝（ｒ_{0\,i}，ｔ_{1\,i}$

，e_{1i}，c_{0i}）を生成する。$Ch_i=1$の場合、証明者アルゴリズムPは、$Rsp_i=(r_{1i}, t_{0i}, e_{0i}, c_{2i})$を生成する。$Ch_i=2$の場合、証明者アルゴリズムPは、$Rsp_i=(r_{1i}, t_{1i}, e_{1i}, c_{1i})$を生成する。

【００９２】

工程＃３で生成された返答Rsp_1，…，Rsp_Nは、検証者アルゴリズムVに送られる。

【００９３】

工程＃４：

返答Rsp_1，…，Rsp_Nを受け取った検証者アルゴリズムVは、受け取った返答Rsp_1，…，Rsp_Nを利用して以下の処理（１）～処理（３）をi＝１～Nについて実行する。但し、検証者アルゴリズムVは、$Ch_i=0$の場合に処理（１）を実行し、$Ch_i=1$の場合に処理（２）を実行し、$Ch_i=2$の場合に処理（３）を実行する。

【００９４】

処理（１）：$Ch_i=0$の場合、検証者アルゴリズムVは、Rsp_iから（r_{0i}，t_{1i}，e_{1i}，c_{0i}）を取り出す。次いで、検証者アルゴリズムVは、$c_{1i}=H(r_{0i}-t_{1i}, F(r_{0i})-e_{1i})$を計算する。さらに、検証者アルゴリズムVは、$c_{2i}=H(t_{1i}, e_{1i})$を計算する。そして、検証者アルゴリズムVは、（$c_{0i}$，$c_{1i}$，$c_{2i}$）を保持する。

【００９５】

処理（２）：$Ch_i=1$の場合、検証者アルゴリズムVは、Rsp_iから（r_{1i}，t_{0i}，e_{0i}，c_{2i}）を取り出す。次いで、検証者アルゴリズムVは、$c_{0i}=H(r_{1i}, G(t_{0i}, r_{1i})+e_{0i})$を計算する。さらに、検証者アルゴリズムVは、$c_{1i}=H(t_{0i}, e_{0i})$を計算する。そして、検証者アルゴリズムVは、（$c_{0i}$，$c_{1i}$，$c_{2i}$）を保持する。

【００９６】

処理（３）：$Ch_i=2$の場合、検証者アルゴリズムVは、Rsp_iから（r_{1i}，t_{1i}，e_{1i}，c_{1i}）を取り出す。次いで、検証者アルゴリズムVは、$c_{0i}=H(r_{1i}, y-F(r_{1i})-G(t_{1i}, r_{1i})-e_{1i})$を計算する。さらに、検証者アルゴリズムVは、$c_{2i}=H(t_{1i}, e_{1i})$を計算する。そして、検証者アルゴリズムVは、（$c_{0i}$，$c_{1i}$，$c_{2i}$）を保持する。

【００９７】

処理（１）～処理（３）をi＝１～Nについて実行した後、検証者アルゴリズムVは、$Cmt=H(c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N})$の等号が成り立つか否かを検証する。検証者アルゴリズムVは、この検証が成功した場合に認証成功を示す値１を出力し、検証に失敗した場合に認証失敗を示す値０を出力する。

【００９８】

以上、３パス方式に係る効率的な並列化アルゴリズムの構成例について説明した。

【００９９】

［２－３：３次多変数多項式に基づくアルゴリズムの構成例］

次に、下記の式（１２）で表現される環R上の３次多項式f_lを利用した効率的なアルゴリズムの構築方法について検討してみたい。３次多項式f_lの組で表現される多変数多項式$F=(f_1, \cdots, f_m)$は、下記の式（１３）の関係を満たす。但し、$G_x(x, y)$は、xに関して１次となる項を表す。また、$G_y(x, y)$は、yに関して１次となる項を表す。$G_x=(g_{x1}, \cdots, g_{xm})$、$G_y=(g_{y1}, \cdots, g_{ym})$と表現すると、$g_{xl}$及び$g_{yl}$は、それぞれ下記の式（１４）及び式（１５）のように展開することができる。但し、g_{xl}の右辺第２項は、x及びyのいずれに関しても１次であるため、g_{yl}に含めてもよい。

【０１００】

【数１０】

$$f_l(x_1,\cdots,x_n)=\sum_{i,j,k}a_{lijk}x_ix_jx_k+\sum_{i,j}b_{lij}x_ix_j+\sum_i c_{li}x_i$$

…（１２）

$$F(x+y)-F(x)-F(y)=G_x(x,y)+G_y(x,y)$$

…（１３）

$$g_{xl}(x_1,\cdots,x_n,y_1,\cdots,y_n)=\sum_{i,j,k}\left(a_{lijk}+a_{likj}+a_{lkji}\right)y_iy_jx_k+\sum_{i,j}\left(b_{lij}+b_{lji}\right)x_iy_j$$

…（１４）

$$g_{yl}(x_1,\cdots,x_n,y_1,\cdots,y_n)=\sum_{i,j,k}\left(a_{lijk}+a_{likj}+a_{lkji}\right)x_ix_jy_k$$

…（１５）

【０１０１】

上記の式（１４）及び式（１５）から分かるように、G_x（ｘ，ｙ）及びG_y（ｘ，ｙ）は、ｘ及びｙについて加法準同型になっている。そこで、これらの性質を利用し、２次多項式f_iを利用した効率的なアルゴリズムの構築方法と同様に、新たな変数r_0，r_1，t_0，u_0，e_0を導入して公開鍵Ｆ（ｓ）を分割する。

【０１０２】

多項式G_x，G_yが加法準同型であるから、変数r_0，r_1，t_0，u_0，e_0を用いて、下記の式（１６）～式（１９）の関係が成り立つ。下記の式（１６）～式（１９）は、（r_0，t_0，u_0，e_0）で再現可能な第１の部分と、（r_0，u_1，e_1）で再現可能な第２の部分と、（r_1，t_0，e_0）で再現可能な第３の部分と、（r_1，t_1，u_1，e_1）で再現可能な第４の部分と、に分けることができる。

【０１０３】

例えば、下記の式（１７）に含まれる「r_0，t_0」、下記の式（１８）に含まれる「u_0」、下記の式（１９）に含まれる「Ｆ（r_0），G_y（r_0，u_0），e_0」、は、第１の部分である。また、下記の式（１９）に含まれる「G_y（r_0，u_1），e_1」は、第２の部分である。さらに、下記の式（１６）に含まれる「e_0，G_x（r_0，r_1）」は、第３の部分である。そして、下記の式（１６）に含まれる「e_1，Ｆ（r_1），G_x（t_1，r_1）」、下記の式（１７）に含まれる「t_1」、下記の式（１８）に含まれる「u_1」は、第４の部分である。

【０１０４】

言い換えると、下記の式（１６）は第３及び第４の部分で構成され、下記の式（１７）及び下記の式（１８）は第１及び第４の部分で構成され、下記の式（１９）は第１及び第２の部分で構成されている。このように、下記の式（１６）～式（１９）は、それぞれ２種類の部分で構成されている。

【０１０５】
また、秘密鍵ｓの定義及び下記の式（１６）～式（１９）の関係から、（r_0，t_0，u_0，e_0）、（r_0，u_1，e_1）、（r_1，t_0，e_0）、（r_1，t_1，u_1，e_1）のいずれか１つを用いても秘密鍵ｓは得られないことが保証される。これらの性質を利用すると、例えば、環Ｒ上の３次多項式f_1を利用した効率的なアルゴリズム（以下、拡張アルゴリズム）を構築することができる。
【０１０６】
【数１１】

$$F(r_0+r_1)=e_0+e_1+F(r_1)+G_x(t_0,r_1)+G_x(t_1,r_1)$$
…（１６）

$$r_0=t_0+t_1$$
…（１７）

$$r_1=u_0+u_1$$
…（１８）

$$F(r_0)+G_y(r_0,u_1)+G_y(r_0,u_0)=e_0+e_1$$
…（１９）

【０１０７】
以下、具体的な拡張アルゴリズムの構成例について説明する。拡張アルゴリズムの設計に関する基本的なポイントは、下記の式（２０）～式（２２）で表現されるメッセージを検証者に送ること、及び第１～第４の部分のいずれかに関する検証を行うことの２点である。但し、この検証だけでは、第３の部分に含まれる「r_1」と第４の部分に含まれる「r_1」とが等しいかを検証することができない。同様に、第１の部分に含まれる「r_0」と第２の部分に含まれる「r_0」とが等しいか、第１の部分に含まれる「t_0，e_0」と第３の部分に含まれる「t_0，e_0」とが等しいかも検証することができない。さらに、第２の部分に含まれる「u_1，e_1」と第４の部分に含まれる「u_1，e_1」とが等しいかも検証することができない。そこで、以下では、これらの検証も可能にした構成例を紹介する。
【０１０８】

【数１２】

$$c_0 = H(G_x(t_0, r_1) + e_0)$$

…（２０）

$$c_1 = H(t_0, u_0)$$

…（２１）

$$c_2 = H(e_1 - G_y(r_0, u_1))$$

…（２２）

【０１０９】

以下、図６を参照しながら、３パス方式に係る拡張アルゴリズムの構成例について説明する。なお、鍵生成アルゴリズムＧｅｎの構成については説明を省略する。

【０１１０】

工程＃１：

図６に示すように、証明者アルゴリズムＰは、ランダムにベクトル$ｒ_0$，$ｔ_0$，$ｕ_0 \in K^n$及び$ｅ_0 \in K^m$を生成する。次いで、証明者アルゴリズムＰは、$ｒ_1 \leftarrow ｓ - ｒ_0$を計算する。この計算は、秘密鍵ｓをベクトル$ｒ_0$によりマスクする操作に相当する。次いで、証明者アルゴリズムＰは、$ｔ_1 \leftarrow ｒ_0 - ｔ_0$を計算する。次いで、証明者アルゴリズムＰは、$ｕ_1 \leftarrow ｒ_1 - ｕ_0$を計算する。次いで、証明者アルゴリズムＰは、$ｅ_1 \leftarrow Ｆ（ｒ_0）- ｅ_0$を計算する。

【０１１１】

工程＃１（続き）：

次いで、証明者アルゴリズムＰは、$ｃ_0 \leftarrow Ｈ（ｒ_1，Ｇ_x（ｔ_0，ｒ_1）＋ｅ_0）$を計算する。次いで、証明者アルゴリズムＰは、$ｃ_1 \leftarrow Ｈ（ｒ_0 - ｔ_0，ｕ_0）$を計算する。次いで、証明者アルゴリズムＰは、$ｃ_2 \leftarrow Ｈ（ｒ_0，ｅ_1 - Ｇ_y（ｒ_0，ｕ_1））$を計算する。次いで、証明者アルゴリズムＰは、$ｃ_3 \leftarrow Ｈ（ｔ_0，ｅ_0）$を計算する。次いで、証明者アルゴリズムＰは、$ｃ_4 \leftarrow Ｈ（ｕ_1，ｅ_1）$を計算する。工程＃１で生成されたメッセージ（$ｃ_0$，$ｃ_1$，$ｃ_2$，$ｃ_3$，$ｃ_4$）は、検証者アルゴリズムＶに送られる。

【０１１２】

工程＃２：

メッセージ（$ｃ_0$，$ｃ_1$，$ｃ_2$，$ｃ_3$，$ｃ_4$）を受け取った検証者アルゴリズムＶは、４つの検証パターンのうち、どの検証パターンを利用するかを選択する。例えば、検証者アルゴリズムＶは、検証パターンの種類を表す４つの数値｛０，１，２，３｝の中から１つの数値を選択し、選択した数値を要求Ｃｈに設定する。この要求Ｃｈは証明者アルゴリズムＰに送られる。

【０１１３】

工程＃３：

要求Ｃｈを受け取った証明者アルゴリズムＰは、受け取った要求Ｃｈに応じて検証者アルゴリズムＶに送る返答Ｒｓｐを生成する。Ｃｈ＝０の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（$ｒ_0$，$ｔ_0$，$ｕ_0$，$ｅ_0$）を生成する。Ｃｈ＝１の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（$ｒ_0$，$ｕ_1$，$ｅ_1$）を生成する。Ｃｈ＝２の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（$ｒ_1$，$ｔ_0$，$ｅ_0$）を生成する。Ｃｈ＝３の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝（$ｒ_1$，$ｔ_1$，$ｕ_1$，$ｅ_1$）を生成する。工程＃３で生成された返答Ｒｓｐは、検証者アルゴリズムＶに送られる。

【０１１４】
工程＃４：
返答Ｒｓｐを受け取った検証者アルゴリズムＶは、受け取った返答Ｒｓｐを利用して以下の検証処理を実行する。
【０１１５】
Ｃｈ＝０の場合、検証者アルゴリズムＶは、$c_1=H(r_0-t_0, u_0)$の等号が成り立つか否かを検証する。次いで、検証者アルゴリズムＶは、$c_2=H(r_0, F(r_0)+G_y(r_0, u_0)-e_0)$の等号が成り立つか否かを検証する。次いで、検証者アルゴリズムＶは、$c_3=H(t_0, e_0)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１１６】
Ｃｈ＝１の場合、検証者アルゴリズムＶは、$c_2=H(r_0, e_1-G_y(r_0, u_1))$の等号が成り立つか否かを検証する。次いで、検証者アルゴリズムＶは、$c_4=H(u_1, e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１１７】
Ｃｈ＝２の場合、検証者アルゴリズムＶは、$c_0=H(r_1, e_0+G_x(t_0, r_1))$の等号が成り立つか否かを検証する。次いで、検証者アルゴリズムＶは、$c_3=H(t_0, e_0)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１１８】
Ｃｈ＝３の場合、検証者アルゴリズムＶは、$c_0=H(r_1, y-F(r_1)-e_1-G_x(t_1, r_1))$の等号が成り立つか否かを検証する。次いで、検証者アルゴリズムＶは、$c_1=H(t_1, r_1-u_1)$の等号が成り立つか否かを検証する。次いで、検証者アルゴリズムＶは、$c_4=H(u_1, e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１１９】
以上、３パス方式に係る拡張アルゴリズムの構成例について説明した。３次多項式を利用していることにより、より高い安全性が実現されている。
【０１２０】
＜３：５パスの公開鍵認証方式に係るアルゴリズムの構成＞
次に、５パスの公開鍵認証方式に係るアルゴリズムについて説明する。なお、以下の説明において、５パスの公開鍵認証方式のことを「５パス方式」と呼ぶ場合がある。
【０１２１】
３パス方式の場合には対話プロトコル１回当たりの偽証確率が２／３であったが、５パス方式の場合には対話プロトコル１回当たりの偽証確率が１／２＋１／ｑとなる。但し、ｑは、利用する環の位数である。従って、環の位数が十分に大きい場合、５パス方式の方が１回当たりの偽証確率を低減することが可能になり、少ない対話プロトコルの実行回数で、偽証確率を十分に小さくすることができる。
【０１２２】
例えば、偽証確率を$1/2^n$以下にしたい場合、３パス方式においては、ｎ／（ｌｏｇ３－１）＝１．７０１ｎ回以上、対話プロトコルを実行する必要がある。一方、偽証確率を$1/2^n$以下にしたい場合、５パス方式においては、ｎ／（１－ｌｏｇ（１＋１／ｑ））回以上、対話プロトコルを実行する必要がある。従って、ｑ＝２４にすれば、同じセキュリティレベルを実現するのに必要な通信量は、３パス方式に比べ、５パス方式の方が少なくなるのである。

【０１２３】
［３－１：具体的なアルゴリズムの構成例（図７）］
まず、図７を参照しながら、５パス方式に係る具体的なアルゴリズムの構成例について紹介する。図７は、５パス方式に係る具体的なアルゴリズムの構成について説明するための説明図である。ここでは、公開鍵ｐｋの一部として２次多項式の組（$f_1(x)$，…，$f_m(x)$）を利用する場合について考える。但し、２次多項式$f_i(x)$は、上記の式（６）のように表現されるものとする。また、ベクトル（x_1，…，x_n）をｘと表記し、２次多項式の組（$f_1(x)$，…，$f_m(x)$）を多変数多項式Ｆ（ｘ）と表記することにする。
【０１２４】
３パス方式に係るアルゴリズムと同様、２つのベクトル$t_0 \in K^n$、$e_0 \in K^m$を用いて、多変数多項式Ｆ（ｘ＋r_0）をマスクするために用いた多変数多項式$F_1(x)$を$F_1(x)=G(x, t_0)+e_0$のように表現する。この表現を用いると、多変数多項式Ｆ（ｘ＋r_0）について、下記の式（２３）で表現される関係が得られる。
【０１２５】
【数１３】

$$\begin{aligned} &Ch_A \cdot F(x+r_0)+F_1(x) \\ &=Ch_A \cdot F(x)+Ch_A \cdot F(r_0)+Ch_A \cdot G(x,r_0)+G(x,t_0)+e_0 \\ &=Ch_A \cdot F(x)+G(x,Ch_A \cdot r_0+t_0)+Ch_A \cdot F(r_0)+e_0 \end{aligned}$$

…（２３）

【０１２６】
そのため、$t_1=Ch_A \cdot r_0+t_0$、$e_1=Ch_A \cdot F(r_0)+e_0$とすれば、マスク後の多変数多項式$F_2(x)=Ch_A \cdot F(x+r_0)+F_1(x)$も、２つのベクトル$t_1 \in K^n$、$e_1 \in K^m$により表現することができる。これらの理由から、$F_1(x)=G(x, t_0)+e_0$と設定すれば、$K^n$上のベクトル及び$K^m$上のベクトルを用いて$F_1$及び$F_2$を表現できるようになり、通信に必要なデータサイズが少ない効率的なアルゴリズムを実現することが可能になる。
【０１２７】
なお、F_2（或いはF_1）からr_0に関する情報が一切漏れることはない。例えば、e_1及びt_1（或いはe_0及びt_0）を与えられても、e_0及びt_0（或いはe_1及びt_1）を知らない限り、r_0の情報を一切知ることはできない。従って、零知識性は担保される。以下、上記の論理に基づいて構築された５パス方式のアルゴリズムについて説明する。ここで説明する５パス方式のアルゴリズムは、以下のような鍵生成アルゴリズムＧｅｎ、証明者アルゴリズムＰ、検証者アルゴリズムＶにより構成される。
【０１２８】
（鍵生成アルゴリズムＧｅｎ）
鍵生成アルゴリズムＧｅｎは、環Ｋ上で定義される多変数多項式$f_1(x_1, \cdots, x_n)$，…，$f_m(x_1, \cdots, x_n)$、及びベクトル$s=(s_1, \cdots, s_n) \in K^n$を生成する。次に、鍵生成アルゴリズムＧｅｎは、$y=(y_1, \cdots, y_m) \leftarrow (f_1(s), \cdots, f_m(s))$を計算する。そして、鍵生成アルゴリズムＧｅｎは、（f_1，…，f_m，ｙ）を公開鍵ｐｋに設定し、ｓを秘密鍵に設定する。なお、以下では、ベクトル（x_1，…，x_n）をｘと表記し、多変数多項式の組（$f_1(x)$，…，$f_m(x)$）をＦ（ｘ）と表記する。
【０１２９】
（証明者アルゴリズムＰ、検証者アルゴリズムＶ）
以下、図７を参照しながら、対話プロトコルの中で証明者アルゴリズムＰ及び検証者ア

ルゴリズムＶにより実行される処理について説明する。この対話プロトコルの中で、証明者は、秘密鍵ｓの情報を検証者に一切漏らさずに、「自身がｙ＝Ｆ（ｓ）を満たすｓを知っていること」を検証者に示す。一方、検証者は、証明者がｙ＝Ｆ（ｓ）を満たすｓを知っているか否かを検証する。なお、公開鍵ｐｋは、検証者に公開されているものとする。また、秘密鍵ｓは、証明者により秘密に管理されているものとする。以下、図７に示したフローチャートに沿って説明を進める。

【０１３０】

工程＃１：

図７に示すように、まず、証明者アルゴリズムＰは、ランダムにベクトル$r_0 \in K^n$、$t_0 \in K^n$、$e_0 \in K^m$を生成する。次いで、証明者アルゴリズムＰは、$r_1 \leftarrow s - r_0$を計算する。この計算は、秘密鍵ｓをベクトルr_0によりマスクする操作に相当する。次いで、証明者アルゴリズムＰは、ベクトルr_0，t_0，e_0のハッシュ値c_0を生成する。つまり、証明者アルゴリズムＰは、$c_0 \leftarrow H(r_0, t_0, e_0)$を計算する。次いで、証明者アルゴリズムＰは、$G(t_0, r_1) + e_0$及び$r_1$のハッシュ値$c_1$を生成する。つまり、証明者アルゴリズムＰは、$c_0 \leftarrow H(r_1, G(t_0, r_1) + e_0)$を計算する。工程＃１で生成されたメッセージ（$c_0$，$c_1$）は、検証者アルゴリズムＶに送られる。

【０１３１】

工程＃２：

メッセージ（c_0，c_1）を受け取った検証者アルゴリズムＶは、ｑ通り存在する環Ｋの元からランダムに１つの数Ch_Aを選択し、選択した数Ch_Aを証明者アルゴリズムＰに送る。

【０１３２】

工程＃３：

数Ch_Aを受け取った証明者アルゴリズムＰは、$t_1 \leftarrow Ch_A \cdot r_0 - t_0$を計算する。さらに、証明者アルゴリズムＰは、$e_1 \leftarrow Ch_A \cdot F(r_0) - e_0$を計算する。そして、証明者アルゴリズムＰは、$t_1$及び$e_1$を検証者アルゴリズムＶに送る。

【０１３３】

工程＃４：

t_1及びe_1を受け取った検証者アルゴリズムＶは、２つの検証パターンのうち、どちらの検証パターンを利用するかを選択する。例えば、検証者アルゴリズムＶは、検証パターンの種類を表す２つの数値｛０，１｝の中から１つの数値を選択し、選択した数値を要求Ch_Bに設定する。この要求Ch_Bは証明者アルゴリズムＰに送られる。

【０１３４】

工程＃５：

要求Ch_Bを受け取った証明者アルゴリズムＰは、受け取った要求Ch_Bに応じて検証者アルゴリズムＶに送り返す返答Ｒｓｐを生成する。$Ch_B = 0$の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝r_0を生成する。$Ch_B = 1$の場合、証明者アルゴリズムＰは、返答Ｒｓｐ＝r_1を生成する。工程＃５で生成された返答Ｒｓｐは、検証者アルゴリズムＶに送られる。

【０１３５】

工程＃６：

返答Ｒｓｐを受け取った検証者アルゴリズムＶは、受け取った返答Ｒｓｐを利用して以下の検証処理を実行する。

【０１３６】

$Ch_B = 0$の場合、検証者アルゴリズムＶは、$r_0 \leftarrow$Ｒｓｐを実行する。そして、検証者アルゴリズムＶは、$c_0 = H(r_0, Ch_A \cdot r_0 - t_1, Ch_A \cdot F(r_0) - e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、この検証が成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。

【０１３７】
$Ch_B=1$の場合、検証者アルゴリズムＶは、$r_1 \leftarrow Rsp$を実行する。そして、検証者アルゴリズムＶは、$c_1=H_1(r_1, Ch_A\cdot(y-F(r_1))-G(t_1, r_1)-e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、この検証が成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１３８】
以上、５パス方式に係る効率的なアルゴリズムの構成例について説明した。
【０１３９】
［３－２：並列化アルゴリズムの構成例（図８）］
次に、図８を参照しながら、図７に示した５パス方式のアルゴリズムを並列化する方法について説明する。なお、鍵生成アルゴリズムＧｅｎの構成については説明を省略する。
【０１４０】
先に述べた通り、５パス方式に係る対話プロトコルを適用すれば、偽証が成功する確率を（１／２＋１／ｑ）以下に抑制することができる。従って、この対話プロトコルを２回実行すれば、偽証が成功する確率を$(1/2+1/q)^2$以下に抑制することができる。さらに、この対話プロトコルをＮ回実行すると、偽証が成功する確率は$(1/2+1/q)^N$となり、Ｎを十分に大きい数（例えば、Ｎ＝８０）にすれば、偽証が成功する確率は無視できる程度に小さくなる。
【０１４１】
対話プロトコルを複数回実行する方法としては、例えば、メッセージ、要求、返答のやり取りを逐次的に複数回繰り返す直列的な方法と、１回分のやり取りで複数回分のメッセージ、要求、返答のやり取りを行う並列的な方法とが考えられる。さらに、直列的な方法と並列的な方法とを組み合わせたハイブリッド型の方法も考えられる。ここでは、５パス方式に係る上記の対話プロトコルを並列的に実行するアルゴリズム（以下、並列化アルゴリズム）について説明する。
【０１４２】
工程＃１：
図８に示すように、まず、証明者アルゴリズムＰは、ｉ＝１～Ｎについて処理（１）～処理（４）を実行する。
処理（１）：証明者アルゴリズムＰは、ランダムにベクトル$r_{0i}, t_{0i} \in K^n$及び$e_{0i} \in K^m$を生成する。
処理（２）：証明者アルゴリズムＰは、$r_{1i} \leftarrow s-r_{0i}$を計算する。この計算は、秘密鍵ｓをベクトル$r_{0i}$によりマスクする操作に相当する。
処理（３）：証明者アルゴリズムＰは、$c_{0i} \leftarrow H(r_{0i}, t_{0i}, e_{0i})$を計算する。
処理（４）：証明者アルゴリズムＰは、$c_{1i} \leftarrow H(r_{1i}, G(t_{0i}, r_{1i})+e_{0i})$を計算する。
ｉ＝１～Ｎについて処理（１）～処理（４）を実行した後、証明者アルゴリズムＰは、ハッシュ値$Cmt \leftarrow H(c_{01}, c_{11}, \cdots, c_{0N}, c_{1N})$を実行する。そして、工程＃１で生成されたハッシュ値Ｃｍｔは、検証者アルゴリズムＶに送られる。
【０１４３】
工程＃２：
ハッシュ値Ｃｍｔを受け取った検証者アルゴリズムＶは、ｉ＝１～Ｎのそれぞれについて、ｑ通り存在する環Ｋの元からランダムに１つの数Ch_{Ai}を選択し、選択した数Ch_{Ai}（ｉ＝１～Ｎ）を証明者アルゴリズムＰに送る。
【０１４４】
工程＃３：
数Ch_{Ai}（ｉ＝１～Ｎ）を受け取った証明者アルゴリズムＰは、ｉ＝１～Ｎのそれぞれについて、$t_{1i} \leftarrow Ch_{Ai}\cdot r_{0i}-t_{0i}$を計算する。さらに、証明者アルゴリズ

ムＰは、ｉ＝１～Ｎのそれぞれについて、e_{1i}←Ch_{Ai}・Ｆ（r_{0i}）－e_{0i}を計算する。次いで、証明者アルゴリズムＰは、ハッシュ値ｄ←Ｈ（t_{11}，e_{11}，…，t_{1N}，e_{1N}）を計算する。そして、証明者アルゴリズムＰは、ハッシュ値ｄを検証者アルゴリズムＶに送る。

【０１４５】

工程＃４：

ハッシュ値ｄを受け取った検証者アルゴリズムＶは、ｉ＝１～Ｎのそれぞれについて、２つの検証パターンのうち、どちらの検証パターンを利用するかを選択する。例えば、検証者アルゴリズムＶは、検証パターンの種類を表す２つの数値｛０，１｝の中から１つの数値を選択し、選択した数値を要求Ch_{Bi}に設定する。要求Ch_{Bi}（ｉ＝１～Ｎ）は証明者アルゴリズムＰに送られる。

【０１４６】

工程＃５：

要求Ch_{Bi}（ｉ＝１～Ｎ）を受け取った証明者アルゴリズムＰは、ｉ＝１～Ｎについて、受け取った要求Ch_{Bi}に応じて検証者アルゴリズムＶに送り返す返答Rsp_iを生成する。Ch_{Bi}＝０の場合、証明者アルゴリズムＰは、返答Rsp_i＝（r_{0i}，t_{0i}，e_{0i}，c_{1i}）を生成する。Ch_{Bi}＝１の場合、証明者アルゴリズムＰは、返答Rsp_i＝（r_{1i}，t_{1i}，e_{1i}，c_{0i}）を生成する。工程＃５で生成された返答Rsp_i（ｉ＝１～Ｎ）は、検証者アルゴリズムＶに送られる。

【０１４７】

工程＃６：

返答Rsp_i（ｉ＝１～Ｎ）を受け取った検証者アルゴリズムＶは、受け取った返答Rsp_i（ｉ＝１～Ｎ）を利用して以下の処理（１）及び処理（２）を実行する。

【０１４８】

処理（１）：Ch_{Bi}＝０の場合、検証者アルゴリズムＶは、（r_{0i}，t_{0i}，e_{0i}，c_{1i}）←Rsp_iを実行する。そして、検証者アルゴリズムＶは、c_{0i}＝Ｈ（r_{0i}，t_{0i}，e_{0i}）を計算する。さらに、検証者アルゴリズムＶは、t_{1i}←Ch_{Ai}・r_{0i}＋t_{0i}、及びe_{1i}←Ch_{Ai}・Ｆ（r_{0i}）－e_{0i}を計算する。そして、検証者アルゴリズムＶは、（c_{0i}，c_{1i}，t_{1i}，e_{1i}）を保持する。

【０１４９】

処理（２）：Ch_{Bi}＝１の場合、検証者アルゴリズムＶは、（r_{1i}，t_{1i}，e_{1i}，c_{0i}）←Rsp_iを実行する。そして、検証者アルゴリズムＶは、c_{1i}＝Ｈ（r_{1i}，Ch_{Ai}・（ｙ－Ｆ（r_{1i}））－Ｇ（t_{1i}，r_{1i}）－e_{1i}）を計算する。そして、検証者アルゴリズムＶは、（c_{0i}，c_{1i}，t_{1i}，e_{1i}）を保持する。

【０１５０】

ｉ＝１～Ｎについて処理（１）及び処理（２）を実行した後、検証者アルゴリズムＶは、Ｃｍｔ＝Ｈ（c_{01}，c_{11}，…，c_{0N}，c_{1N}）の等号が成り立つか否かを検証する。さらに、検証者アルゴリズムＶは、ｄ＝Ｈ（t_{11}，e_{11}，…，t_{1N}，e_{1N}）の等号が成り立つか否かを検証する。そして、検証者アルゴリズムＶは、これらの検証が全て成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。

【０１５１】

以上、５パス方式に係る効率的な並列化アルゴリズムの構成例について説明した。

【０１５２】

［３－３：３次多変数多項式に基づくアルゴリズムの構成例］

さて、３パス方式の場合と同様、環Ｒ上の３次多項式f_lを利用した効率的なアルゴリズムの構築方法について検討してみたい。３次多項式f_lを上記の式（１２）のように表現した場合に、G_x（ｘ，ｙ）及びG_y（ｘ，ｙ）がｘ及びｙについて線形になることについては式（１４）と式（１５）より分かる。

【０１５３】

そこで、上記の性質を利用し、新たな変数r_0，r_1，t_0，u_0，e_0を導入して、公開鍵Ｆ（ｓ）をCh_A倍した項を分割する。多項式G_x，G_yがそれぞれｘとｙについて線形であるから、変数r_0，r_1，t_0，u_0，e_0を用いて、下記の式（２４）～式（２７）の関係が成り立つ。また、下記の式（２４）～式（２７）は、Ch_Aに依存する第１の部分と、Ch_Aに依存しない第２の部分とに分けることができる。但し、第１の部分は、（r_1，t_1，u_1，e_1）で再現可能である。また、第２の部分は、（r_0，t_1，u_1，e_1）で再現可能である。

【０１５４】

例えば、下記の式（３２）に含まれる「e_0，G_x（t_0，r_1）」、下記の式（３３）に含まれる「t_0」、下記の式（２６）に含まれる「u_0」、下記の式（２７）に含まれる「e_0，G_y（r_0，u_0）」は第１の部分である。一方、下記の式（２４）に含まれる「$Ch_A \cdot F(r_0+r_1)$，e_1，$Ch_A \cdot F(r_1)$，$G_x(t_1,r_1)$」、下記の式（２５）に含まれる「$Ch_A \cdot r_0$，t_1」、下記の式（３４）に含まれる「$Ch_A \cdot r_1$，u_1」、下記の式（２７）に含まれる「$Ch_A \cdot F(r_0)$，$G_y(r_0,u_1)$，e_1」は第２の部分である。

【０１５５】

また、秘密鍵ｓの定義及び下記の式（２４）～式（２７）の関係から、（r_1，t_1，u_1，e_1）、（r_0，t_1，u_1，e_1）のいずれか１つを用いても秘密鍵ｓは得られないことが保証される。これらの性質を利用すると、例えば、環Ｒ上の３次多項式f_1を利用した効率的なアルゴリズム（以下、拡張アルゴリズム）を構築することができる。

【０１５６】

【数１４】

$$Ch_A \cdot F(r_0+r_1) = e_0 + e_1 + Ch_A \cdot F(r_1) + G_x(t_0,r_1) + G_x(t_1,r_1)$$

…（２４）

$$Ch_A \cdot r_0 = t_0 + t_1$$

…（２５）

$$Ch_A \cdot r_1 = u_0 + u_1$$

…（２６）

$$Ch_A \cdot F(r_0) + G_y(r_0,u_1) + G_y(r_0,u_0) = e_0 + e_1$$

…（２７）

【０１５７】

以下、具体的な拡張アルゴリズムの構成例について説明する。拡張アルゴリズムの設計に関する基本的なポイントは、下記の式（２８）及び式（２９）で表現されるメッセージを検証者に送ること、及び検証者が選択したCh_Aについて、Ch_Aに依存する部分（第１の部分）に関する検証を行うことの２点である。但し、「メッセージを生成する際に利用したr_0及びr_1を、検証の際に別のr_0及びr_1にすり替えられることを防ぐ」ために、r_0及びr_1に関する検証を追加した構成例を以下で紹介する。

【０１５８】

【数１５】

$$c_0 = H(t_0, e_0 - G_y(r_0, u_0))$$

…（２８）

$$c_1 = H(u_0, G_x(t_0, r_1) + e_0)$$

…（２９）

【０１５９】

以下、図９を参照しながら、５パス方式に係る拡張アルゴリズムの基本構成について説明する。なお、鍵生成アルゴリズムＧｅｎの構成については説明を省略する。

【０１６０】

工程＃１：

図９に示すように、証明者アルゴリズムＰは、ランダムにベクトルr_0，t_0，$u_0 \in K^n$及び$e_0 \in K^m$を生成する。次いで、証明者アルゴリズムＰは、$r_1 \leftarrow s - r_0$を計算する。この計算は、秘密鍵ｓをベクトルr_0によりマスクする操作に相当する。次いで、証明者アルゴリズムＰは、$c_0 \leftarrow H(r_0, t_0, e_0 - G_y(r_0, u_0))$を計算する。次いで、証明者アルゴリズムＰは、$c_1 \leftarrow H(r_1, u_0, G_x(t_0, r_1) + e_0)$を計算する。工程＃１で生成されたメッセージ（$c_0$，$c_1$）は、検証者アルゴリズムＶに送られる。

【０１６１】

工程＃２：

メッセージ（c_0，c_1）を受け取った検証者アルゴリズムＶは、ランダムに数Ch_Aを選択する。この数Ch_Aは証明者アルゴリズムＰに送られる。

【０１６２】

工程＃３：

数Ch_Aを受け取った証明者アルゴリズムＰは、$t_1 \leftarrow Ch_A \cdot r_0 - t_0$を計算する。次いで、証明者アルゴリズムＰは、$u_1 \leftarrow Ch_A \cdot r_1 - u_0$を計算する。次いで、証明者アルゴリズムＰは、$e_1 \leftarrow Ch_A \cdot F(r_0) + Ch_A \cdot G_y(r_0, r_1) - e_0$を計算する。工程＃３で生成された（$t_1$，$u_1$，$e_1$）は、検証者アルゴリズムＶに送られる。

【０１６３】

工程＃４：

（t_1，u_1，e_1）を受け取った検証者アルゴリズムＶは、２つの検証パターンのうち、どの検証パターンを利用するかを選択する。例えば、検証者アルゴリズムＶは、検証パターンの種類を表す２つの数値｛０，１｝の中から１つの数値を選択し、選択した数値を要求Ch_Bに設定する。この要求Ch_Bは証明者アルゴリズムＰに送られる。

【０１６４】

工程＃５：

要求Ch_Bを受け取った証明者アルゴリズムＰは、受け取った要求Ch_Bに応じて検証者アルゴリズムＶに送る返答Ｒｓｐを生成する。$Ch_B = 0$の場合、証明者アルゴリズムＰは、返答$Rsp = r_0$を生成する。$Ch_B = 1$の場合、証明者アルゴリズムＰは、返答$Rsp = r_1$を生成する。工程＃５で生成された返答Ｒｓｐは、検証者アルゴリズムＶに送られる。

【０１６５】

工程＃６：

返答Ｒｓｐを受け取った検証者アルゴリズムＶは、受け取った返答Ｒｓｐを利用して以下の検証処理を実行する。

【０１６６】
$Ch_B=0$の場合、検証者アルゴリズムＶは、$c_0=H(r_0, Ch_A\cdot r_0-t_1, Ch_A\cdot F(r_0)+G_y(r_0, u_1)-e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、この検証が成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１６７】
$Ch_B=1$の場合、検証者アルゴリズムＶは、$c_1=H(r_1, Ch_A\cdot r_1-u_1, Ch_A\cdot(y-F(r_1))-G_x(t_1, r_1)-e_1)$の等号が成り立つか否かを検証する。検証者アルゴリズムＶは、この検証が成功した場合に認証成功を示す値１を出力し、検証に失敗があった場合に認証失敗を示す値０を出力する。
【０１６８】
以上、５パス方式に係る拡張アルゴリズムの構成例について説明した。３次多項式を利用していることにより、より高い安全性が実現される。
【０１６９】
＜４：電子署名方式への変形＞
次に、上記の公開鍵認証方式を電子署名方式へと変形する方法を紹介する。
【０１７０】
公開鍵認証方式のモデルにおける証明者を電子署名方式における署名者に対応させると、証明者のみが検証者を納得させられるという点において、電子署名方式のモデルと近似していることが容易に理解されよう。こうした考えに基づき、上述した公開鍵認証方式を電子署名方式へと変形する方法について説明する。
【０１７１】
［４－１：３パスの公開鍵認証方式から電子署名方式への変形（図１０）］
まず、３パスの公開鍵認証方式から電子署名方式への変形について説明する。
【０１７２】
図１０に示すように、３パス方式に係る効率的なアルゴリズム（例えば、図５を参照）は、３回の対話及び４つの工程＃１～工程＃４で表現される。
【０１７３】
工程＃１は、$i=1\sim N$について、$a_i=(r_{0i}, t_{0i}, e_{0i}, r_{1i}, t_{1i}, e_{1i}, c_{0i}, c_{1i}, c_{2i})$を生成する処理（１）と、$Cmt\leftarrow H(c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N})$を計算する処理（２）とで構成される。工程＃１で証明者アルゴリズムＰにより生成されたＣｍｔは、検証者アルゴリズムＶへと送られる。
【０１７４】
工程＃２は、$Ch_1, \cdots, Ch_N$を選択する処理で構成される。工程＃２で検証者アルゴリズムＶにより選択された$Ch_1, \cdots, Ch_N$は、証明者アルゴリズムＰへと送られる。
【０１７５】
工程＃３は、$Ch_1, \cdots, Ch_N$及び$a_1, \cdots, a_N$を用いて$Rsp_1, \cdots, Rsp_N$を生成する処理で構成される。この処理を$Rsp_i\leftarrow Select(Ch_i, a_i)$と表現する。工程＃３で証明者アルゴリズムＰにより生成された$Rsp_1, \cdots, Rsp_N$は、検証者アルゴリズムＶへと送られる。
【０１７６】
工程＃４は、$Ch_1, \cdots, Ch_N$及び$Rsp_1, \cdots, Rsp_N$を用いて$c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N}$を再生する処理（１）と、再生した$c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N}$を用いて$Cmt=H(c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N})$を検証する処理（２）とで構成される。
【０１７７】
上記の工程＃１～工程＃４で表現される公開鍵認証方式のアルゴリズムは、図１０に示すような署名生成アルゴリズムＳｉｇ及び署名検証アルゴリズムＶｅｒに変形される。

【０１７８】
（署名生成アルゴリズムＳｉｇ）
まず、署名生成アルゴリズムＳｉｇの構成について述べる。署名生成アルゴリズムＳｉｇは、以下の処理（１）～処理（５）で構成される。
【０１７９】
処理（１）：署名生成アルゴリズムＳｉｇは、$a_i=(r_{0i}, t_{0i}, e_{0i}, r_{1i}, t_{1i}, e_{1i}, c_{0i}, c_{1i}, c_{2i})$を生成する。
処理（２）：署名生成アルゴリズムＳｉｇは、$Cmt \leftarrow H(c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N})$を計算する。
処理（３）：署名生成アルゴリズムＳｉｇは、$(Ch_1, \cdots, Ch_N) \leftarrow H(M, Cmt)$を計算する。このＭは、署名を付与する文書である。
処理（４）：署名生成アルゴリズムＳｉｇは、$Rsp_i \leftarrow Select(Ch_i, a_i)$を計算する。
処理（５）：署名生成アルゴリズムＳｉｇは、$(Cmt, Rsp_1, \cdots, Rsp_N)$を署名に設定する。
【０１８０】
（署名検証アルゴリズムＶｅｒ）
次に、署名検証アルゴリズムＶｅｒの構成について述べる。署名検証アルゴリズムＶｅｒは、以下の処理（１）～処理（３）で構成される。
【０１８１】
処理（１）：署名検証アルゴリズムＶｅｒは、$(Ch_1, \cdots, Ch_N) \leftarrow H(M, Cmt)$を計算する。
処理（２）：署名検証アルゴリズムＶｅｒは、$Ch_1, \cdots, Ch_N$及び$Rsp_1, \cdots, Rsp_N$を用いて$c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N}$を生成する。
処理（３）：署名検証アルゴリズムＶｅｒは、再生した$c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N}$を用いて$Cmt=H(c_{01}, c_{11}, c_{21}, \cdots, c_{0N}, c_{1N}, c_{2N})$を検証する。
【０１８２】
以上説明したように、公開鍵認証方式のモデルにおける証明者を電子署名方式における署名者に対応させることで、公開鍵認証方式のアルゴリズムを電子署名方式のアルゴリズムへと変形することが可能になる。
【０１８３】
［４－２：５パスの公開鍵認証方式から電子署名方式への変形（図１１）］
次に、５パスの公開鍵認証方式から電子署名方式への変形について説明する。
【０１８４】
図１１に示すように、５パス方式に係る効率的なアルゴリズム（例えば、図８を参照）は、５回の対話及び６つの工程＃１～工程＃６で表現される。
【０１８５】
工程＃１は、ｉ＝１～Ｎについて、$a_i=(r_{0i}, t_{0i}, e_{0i}, r_{1i}, t_{1i}, e_{1i}, c_{0i}, c_{1i})$を生成する処理（１）と、$Cmt \leftarrow H(c_{01}, c_{11}, \cdots, c_{0N}, c_{1N})$を計算する処理（２）とで構成される。工程＃１で証明者アルゴリズムＰにより生成されたＣｍｔは、検証者アルゴリズムＶへと送られる。
【０１８６】
工程＃２は、$Ch_{A1}, \cdots, Ch_{AN}$を選択する処理で構成される。工程＃２で検証者アルゴリズムＶにより選択された$Ch_{A1}, \cdots, Ch_{AN}$は、証明者アルゴリズムＰへと送られる。
【０１８７】
工程＃３は、ｉ＝１～Ｎについて、$b_i=(t_{1i}, e_{1i})$を生成する処理、及び$d=H(t_{11}, e_{11}, \cdots, t_{1N}, e_{1N})$を生成する処理で構成される。工程＃３で証明者アルゴリズムＰにより生成されたｄは、検証者アルゴリズムＶへと送られる。

【０１８８】
工程＃４は、Ch_{B1}，…，Ch_{BN}を選択する処理で構成される。工程＃４で検証者アルゴリズムＶにより選択されたCh_{B1}，…，Ch_{BN}は、証明者アルゴリズムＰへと送られる。
【０１８９】
工程＃５は、Ch_{B1}，…，Ch_{BN}，a_1，…，a_N，b_1，…，b_Nを用いてRsp_1，…，Rsp_Nを生成する処理で構成される。この処理を$Rsp_i \leftarrow Select(Ch_{Bi}, a_i, b_i)$と表現する。工程＃５で証明者アルゴリズムＰにより生成されたRsp_1，…，Rsp_Nは、検証者アルゴリズムＶへと送られる。
【０１９０】
工程＃６は、Ch_{A1}，…，Ch_{AN}，Ch_{B1}，…，Ch_{BN}，Rsp_1，…，Rsp_Nを用いてc_{01}，c_{11}，…，c_{0N}，c_{1N}，t_{11}，e_{11}，…，t_{1N}，e_{1N}を再生する処理と、再生したc_{01}，c_{11}，…，c_{0N}，c_{1N}を用いて$Cmt=H(c_{01}, c_{11}, \cdots, c_{0N}, c_{1N})$を検証する処理と、$d=H(t_{11}, e_{11}, \cdots, t_{1N}, e_{1N})$を検証する処理とで構成される。
【０１９１】
上記の工程＃１～工程＃６で表現される公開鍵認証方式のアルゴリズムは、図１１に示すような署名生成アルゴリズムＳｉｇ及び署名検証アルゴリズムＶｅｒに変形される。
【０１９２】
（署名生成アルゴリズムＳｉｇ）
まず、署名生成アルゴリズムＳｉｇの構成について述べる。署名生成アルゴリズムＳｉｇは、以下の処理（１）～処理（７）で構成される。
【０１９３】
処理（１）：署名生成アルゴリズムＳｉｇは、$a_i=(r_{0i}, t_{0i}, e_{0i}, r_{1i}, t_{1i}, e_{1i}, c_{0i}, c_{1i})$を生成する。
処理（２）：署名生成アルゴリズムＳｉｇは、$Cmt \leftarrow H(c_{01}, c_{11}, \cdots, c_{0N}, c_{1N})$を計算する。
処理（３）：署名生成アルゴリズムＳｉｇは、$(Ch_{A1}, \cdots, Ch_{AN}) \leftarrow H(M, Cmt)$を計算する。このＭは、署名を付与する文書である。
処理（４）：署名生成アルゴリズムＳｉｇは、ｉ＝１～Ｎについて、$b_i=(t_{1i}, e_{1i})$を生成する。さらに、署名生成アルゴリズムＳｉｇは、$d=H(t_{11}, e_{11}, \cdots, t_{1N}, e_{1N})$を算出する。
処理（５）：署名生成アルゴリズムＳｉｇは、$(Ch_{B1}, \cdots, Ch_{BN}) \leftarrow H(M, Cmt, Ch_{A1}, \cdots, Ch_{AN}, d)$を計算する。なお、$(Ch_{B1}, \cdots, Ch_{BN}) \leftarrow H(Ch_{A1}, \cdots, Ch_{AN}, d)$と変形してもよい。
処理（６）：署名生成アルゴリズムＳｉｇは、$Rsp_i \leftarrow Select(Ch_{Bi}, a_i, b_i)$を計算する。
処理（７）：署名生成アルゴリズムＳｉｇは、$(Cmt, d, Rsp_1, \cdots, Rsp_N)$を電子署名に設定する。
【０１９４】
（署名検証アルゴリズムＶｅｒ）
次に、署名検証アルゴリズムＶｅｒの構成について述べる。署名検証アルゴリズムＶｅｒは、以下の処理（１）～処理（４）で構成される。
【０１９５】
処理（１）：署名検証アルゴリズムＶｅｒは、$(Ch_{A1}, \cdots, Ch_{AN}) \leftarrow H(M, Cmt)$を計算する。
処理（２）：署名検証アルゴリズムＶｅｒは、$(Ch_{B1}, \cdots, Ch_{BN}) \leftarrow H(M, Cmt, Ch_{A1}, \cdots, Ch_{AN}, d)$を計算する。なお、署名検証アルゴリズムＶｅｒが実行する処理（５）において、$(Ch_{B1}, \cdots, Ch_{BN}) \leftarrow H(Ch_{A1}, \cdots, Ch_{AN}, d)$と変形した場合、署名検証アルゴリズムＶｅｒは、$(Ch_{B1}, \cdots, Ch_{BN}$

）←Ｈ（Ch_{A1}，…，Ch_{AN}，ｄ）を計算する。
処理（３）：署名検証アルゴリズムＶｅｒは、Ch_{A1}，…，Ch_{AN}，Ch_{B1}，…，Ch_{BN}，Rsp_1，…，Rsp_Nを用いてt_{11}，e_{11}，…，t_{1N}，e_{1N}，c_{01}，c_{11}，…，c_{0N}，c_{1N}を生成する。
処理（４）：署名検証アルゴリズムＶｅｒは、再生したc_{01}，c_{11}，…，c_{0N}，c_{1N}を用いてＣｍｔ＝Ｈ（c_{01}，c_{11}，…，c_{0N}，c_{1N}）及びｄ＝Ｈ（t_{11}，e_{11}，…，t_{1N}，e_{1N}，）を検証する。

【０１９６】
以上説明したように、公開鍵認証方式のモデルにおける証明者を電子署名方式における署名者に対応させることで、公開鍵認証方式のアルゴリズムを電子署名方式のアルゴリズムへと変形することが可能になる。

【０１９７】
＜５：署名検証に要するメモリ量の削減方法について＞
ところで、上述した電子署名方式のアルゴリズムにおいては、署名検証アルゴリズムＶｅｒが電子署名全体を受信した後で署名検証の処理を実行していた。しかし、上述した電子署名方式の場合、電子署名のデータサイズが比較的大きい。そのため、ＲＦＩＤ（Ｒａｄｉｏ　Ｆｒｅｑｕｅｎｃｙ　ＩＤｅｎｔｉｆｉｃａｔｉｏｎ）など、少ないメモリ容量しか持たないデバイスを用いて認証を行う場合には、メモリの空き容量や認証処理中のメモリ使用率などに気を配る必要があった。また、十分なメモリ容量を持たないデバイスを利用する場合、認証を行うことができない場合も想定される。そこで、本件発明者は、署名検証に要するメモリ量の削減方法を考案した。

【０１９８】
［５－１：ハッシュ関数の構造について（図１２）］
まず、本件発明者は、ハッシュ関数の構造に注目した。多くの場合、ハッシュ関数は、入力をブロック単位に区切り、ブロック単位で順次処理を進める構造を有している。例えば、ＳＨＡ－１の場合、ハッシュ関数は、図１２に示すような構造を有している。図１２に示すハッシュ関数は、パディング済みの入力ＭをZ個のブロックm_1，…，m_Zに区切り、インデックスｊをインクリメントしながら、逐次的に、ブロックm_jを初期値ＩＶ又は中間値CV_jと共に所定の関数ＣＦに作用させてハッシュ値を生成する。従って、中間値CV_jが得られた時点で、それ以前に利用したブロックは不要になる。そこで、この特性を利用し、アルゴリズムの実行に必要なメモリ量を効果的に削減する仕組み（以下、メモリ削減方法）を考案した。以下、この仕組みを上述した電子署名方式に適用する方法について説明する。

【０１９９】
［５－２：３パス方式に基づく電子署名方式への適用例（図１４）］
まず、図１０に示した３パス方式に基づく電子署名方式のアルゴリズムに上記のメモリ削減方法を適用する方法について説明する。

【０２００】
（通常の実装方法：図１３）
通常、上述した電子署名方式に係る署名検証アルゴリズムＶｅｒは、図１３に示すように、電子署名を構成する（Ｃｍｔ，Rsp_1，…，Rsp_N）を１度に受信する（Ｓ１０１）。次いで、署名検証アルゴリズムＶｅｒは、（Ch_1，…，Ch_N）←Ｈ（Ｍ，Ｃｍｔ）を実行する（Ｓ１０２）。次いで、署名検証アルゴリズムＶｅｒは、（c_{01}，c_{11}，c_{21}，…，c_{0N}，c_{1N}，c_{2N}）←Ｒｅｐｒｏｄｕｃｅ（Ch_1，…，Ch_N；Rsp_1，…，Rsp_N）を実行する（Ｓ１０３）。次いで、署名検証アルゴリズムＶｅｒは、Ｃｍｔ＝Ｈ（c_{01}，c_{11}，c_{21}，…，c_{0N}，c_{1N}，c_{2N}）を検証し（Ｓ１０４）、署名検証に係る一連の処理を終了する。

【０２０１】
（メモリ削減方法：図１４）
通常の実装方法の場合、図１３のステップＳ１０１のように、一度に電子署名を受信す

ると、ステップＳ１０３の処理が完了するまで（Rsp_1，…，Rsp_N）を保持しておくためのメモリが必要になる。しかし、図５のアルゴリズム構成から分かるように、ステップＳ１０３で実行する（c_{0i}，c_{1i}，c_{2i}）の再生には（Ch_i；Rsp_i）以外の情報を利用しない。また、図１２に示したハッシュ関数の構造を考慮すると、ステップＳ１０４において実行されるハッシュ関数の計算はブロック単位で分割して実行できることが分かる。そこで、署名検証に係る処理の構成を図１４に示すような構成に改良する。

【０２０２】

図１４に示す構成の場合、署名検証アルゴリズムＶｅｒは、まず、電子署名に含まれるＣｍｔだけを受信する（Ｓ１１１）。次いで、署名検証アルゴリズムＶｅｒは、（Ch_1，…，Ch_N）←Ｈ（Ｍ，Ｃｍｔ）を実行する（Ｓ１１２）。次いで、署名検証アルゴリズムＶｅｒは、ｉ＝１～Ｎについて、ｉをインクリメントさせながら逐次的にステップＳ１１３～Ｓ１１５の処理を実行する。

【０２０３】

ステップＳ１１３において、署名検証アルゴリズムＶｅｒは、Rsp_iを受信する（Ｓ１１３）。次いで、署名検証アルゴリズムＶｅｒは、受信したRsp_iを用いて、（c_{0i}，c_{1i}，c_{2i}）←Ｒｅｐｒｏｄｕｃｅ（Ch_i；Rsp_i）を実行する（Ｓ１１４）。ステップＳ１１４の処理を実行した後、Ch_i及びRsp_iは不要になる。そこで、署名検証アルゴリズムＶｅｒは、ステップＳ１１４の処理を実行後にCh_i及びRsp_iをメモリから消去する。

【０２０４】

次いで、署名検証アルゴリズムＶｅｒは、$tmp_i \leftarrow H_i(tmp_{i-1}; c_{0i}, c_{1i}, c_{2i})$を実行する（Ｓ１１５）。なお、関数$H_i$は、ハッシュ関数Ｈの内部で$c_{0i}$，$c_{1i}$，$c_{2i}$まで計算した際に生成される中間値を出力する関数である。実際には選択したハッシュ関数に応じて関数H_iの入力サイズは異なるため必要に応じて、ビットを付加するなどの適切な入力長の補正が行われる。関数H_iを用いると、ハッシュ関数Ｈは、以下に示す処理（１）～処理（３）で構成されたアルゴリズムにより表現される。そして、tmp_Nがハッシュ関数Ｈの最終的な出力（ハッシュ値）となる。実際にはハッシュ関数の仕様に応じて、最終処理においてパディングの追加処理を行う。

【０２０５】

処理（１）：tmp_0←空文字列
処理（２）：
ｆｏｒ　ｉ＝１　ｔｏ　Ｎ
　$tmp_i \leftarrow H_i(tmp_{i-1}; c_{0i}, c_{1i}, c_{2i})$
ｅｎｄ　ｆｏｒ
処理（３）：ｏｕｔｐｕｔ　tmp_N

【０２０６】

ｉ＝１～Ｎについて、ステップＳ１１３～Ｓ１１５の処理を実行した後、署名検証アルゴリズムＶｅｒは、Ｃｍｔ＝tmp_Nが成り立つか否かを検証し（Ｓ１１６）、署名検証に係る一連の処理を終了する。上記のように、署名検証アルゴリズムＶｅｒは、ステップＳ１１３～Ｓ１１５の処理を繰り返し実行する過程で不要になった情報をメモリかた消去する。そのため、署名検証に要するメモリ量は最小限に抑えられる。その結果、少ないメモリ容量しか持たないデバイスでも、上記の署名検証が可能になる。

【０２０７】

［５－３：５パス方式に基づく電子署名方式への適用例（図１６）］

次に、図１１に示した５パス方式に基づく電子署名方式のアルゴリズムに上記のメモリ削減方法を適用する方法について説明する。

【０２０８】

（通常の実装方法：図１５）

通常、上述した電子署名方式に係る署名検証アルゴリズムＶｅｒは、図１５に示すよう

に、電子署名を構成する（Ｃｍｔ，ｄ，$Ｒｓｐ_{1}$，…，$Ｒｓｐ_{Ｎ}$）を１度に受信する（Ｓ１２１）。次いで、署名検証アルゴリズムＶｅｒは、（$Ｃｈ_{Ａ１}$，…，$Ｃｈ_{ＡＮ}$）←Ｈ（Ｍ，Ｃｍｔ）を実行する（Ｓ１２２）。次いで、署名検証アルゴリズムＶｅｒは、（$Ｃｈ_{Ｂ１}$，…，$Ｃｈ_{ＢＮ}$）←Ｈ（Ｍ，Ｃｍｔ，$Ｃｈ_{Ａ１}$，…，$Ｃｈ_{ＡＮ}$，ｄ）を実行する（Ｓ１２３）。次いで、署名検証アルゴリズムＶｅｒは、（$ｃ_{０１}$，$ｃ_{１１}$，…，$ｃ_{０Ｎ}$，$ｃ_{１Ｎ}$，$ｄ_{１１}$，$ｅ_{１１}$，…，$ｄ_{１Ｎ}$，$ｅ_{１Ｎ}$）←Ｒｅｐｒｏｄｕｃｅ（$Ｃｈ_{Ａ１}$，…，$Ｃｈ_{ＡＮ}$，$Ｃｈ_{Ｂ１}$，…，$Ｃｈ_{ＢＮ}$；$Ｒｓｐ_{1}$，…，$Ｒｓｐ_{Ｎ}$）を実行する（Ｓ１２４）。次いで、署名検証アルゴリズムＶｅｒは、Ｃｍｔ＝Ｈ（$ｃ_{０１}$，$ｃ_{１１}$，…，$ｃ_{０Ｎ}$，$ｃ_{１Ｎ}$）とｄ＝Ｈ($ｄ_{１１}$，$ｅ_{１１}$，…，$ｄ_{１Ｎ}$，$ｅ_{１Ｎ}$)を検証し（Ｓ１２５）、署名検証に係る一連の処理を終了する。

【０２０９】

（メモリ削減方法：図１６）

図１５のステップＳ１２１のように、一度に電子署名を受信すると、ステップＳ１２４の処理が完了するまで（$Ｒｓｐ_{1}$，…，$Ｒｓｐ_{Ｎ}$）を保持しておくためのメモリが必要になる。しかしながら、図８のアルゴリズム構成から分かるように、ステップＳ１２４で実行する（$ｃ_{０ｉ}$，$ｃ_{１ｉ}$，$ｄ_{１ｉ}$，$ｅ_{１ｉ}$）の再生には（$Ｃｈ_{Ａｉ}$，$Ｃｈ_{Ｂｉ}$；$Ｒｓｐ_{ｉ}$）以外の情報を利用しない。また、図１２に示したハッシュ関数の構造を考慮すると、ステップＳ１２５において実行されるハッシュ関数の計算はブロック単位で分割して実行できることが分かる。そこで、署名検証に係る処理の構成を図１６に示すような構成に改良する。

【０２１０】

図１６に示す構成の場合、署名検証アルゴリズムＶｅｒは、まず、電子署名に含まれるＣｍｔだけを受信する（Ｓ１３１）。次いで、署名検証アルゴリズムＶｅｒは、（$Ｃｈ_{Ａ１}$，…，$Ｃｈ_{ＡＮ}$）←Ｈ（Ｍ，Ｃｍｔ）を実行する（Ｓ１３２）。

【０２１１】

次いで、署名検証アルゴリズムＶｅｒは、ｄを受信する（Ｓ１３３）。次いで、署名検証アルゴリズムＶｅｒは、受信したｄを用いて、（$Ｃｈ_{Ｂ１}$，…，$Ｃｈ_{ＢＮ}$）←Ｈ（Ｍ，Ｃｍｔ，$Ｃｈ_{Ａ１}$，…，$Ｃｈ_{ＡＮ}$，ｄ）を実行する（Ｓ１３４）。ステップＳ１３４の処理を実行した後、ｄは不要になる。そこで、署名検証アルゴリズムＶｅｒは、ステップＳ１３４の処理を実行後にｄをメモリから消去する。次いで、署名検証アルゴリズムＶｅｒは、ｉ＝１～Ｎについて、ｉをインクリメントさせながら逐次的にステップＳ１３５～Ｓ１３７の処理を実行する。

【０２１２】

ステップＳ１３５において、署名検証アルゴリズムＶｅｒは、$Ｒｓｐ_{ｉ}$を受信する（Ｓ１３５）。次いで、署名検証アルゴリズムＶｅｒは、受信した$Ｒｓｐ_{ｉ}$を用いて、（$ｃ_{０ｉ}$，$ｃ_{１ｉ}$，$ｔ_{１ｉ}$，$ｅ_{１ｉ}$）←Ｒｅｐｒｏｄｕｃｅ（$Ｃｈ_{Ａｉ}$，$Ｃｈ_{Ｂｉ}$，$Ｒｓｐ_{ｉ}$）を実行する（Ｓ１３６）。ステップＳ１３６の処理を実行した後、$Ｃｈ_{Ａｉ}$、$Ｃｈ_{Ｂｉ}$及び$Ｒｓｐ_{ｉ}$は不要になる。そこで、署名検証アルゴリズムＶｅｒは、ステップＳ１３６の処理を実行後に$Ｃｈ_{Ａｉ}$、$Ｃｈ_{Ｂｉ}$及び$Ｒｓｐ_{ｉ}$をメモリから消去する。

【０２１３】

次いで、署名検証アルゴリズムＶｅｒは、$ｔｍｐ_{ｉ}$←$Ｈ_{ｉ}$（$ｔｍｐ_{ｉ－１}$；$ｃ_{０ｉ}$，$ｃ_{１ｉ}$）と$ｔｍｐ_{ｉ}$’←$Ｈ_{ｉ}$（$ｔｍｐ_{ｉ－１}$’；$ｔ_{１ｉ}$，$ｅ_{１ｉ}$）を実行する（Ｓ１３７）。ｉ＝１～Ｎについて、ステップＳ１３５～Ｓ１３７の処理を実行した後、署名検証アルゴリズムＶｅｒは、Ｃｍｔ＝$ｔｍｐ_{Ｎ}$とｄ＝$ｔｍｐ_{Ｎ}$’が成り立つか否かを検証し（Ｓ１３８）、署名検証に係る一連の処理を終了する。上記のように、署名検証アルゴリズムＶｅｒは、ステップＳ１３５～Ｓ１３７の処理を繰り返し実行する過程で不要になった情報をメモリから消去する。そのため、署名検証に要するメモリ量は最小限に抑えられる。その結果、少ないメモリ容量しか持たないデバイスでも、上記の署名検証が可能になる。

【０２１４】

以上、署名検証に要するメモリ量の削減方法について説明した。

【０２１５】
＜６：２進数の乱数列から３進数の乱数列を抽出する方法について＞
ところで、３パス方式に基づく公開鍵認証方式のアルゴリズムにおいて、３進数の一様乱数をＮ個以上生成する場面が存在する。しかし、３進数の一様乱数を生成する優れた乱数生成器は一般的ではない。そのため、２進数の一様乱数を生成する優れた乱数生成器を用いて３進数の一様乱数を生成する方法を考える必要がある。そこで、本件発明者は、２進数の一様乱数から３進数の一様乱数を効率的に生成する方法を考案した。以下、この方法について詳細に説明する。なお、以下の説明において、１進表現（１は２又は３）で表された１つの数を１シンボルと数えることにする。
【０２１６】
［６－１：抽出方法＃１（２ビット区切り）（図１７）］
まず、図１７を参照しながら、Ｍビットの２進数を２ビットずつ区切って３進数を抽出する方法（以下、抽出方法＃１）を紹介する。図１７に示すように、２進表現の乱数列を２ビットずつ区切ると、Ｍ／２個の２ビットの乱数が得られる。例えば、「００」を３進数の「０」、「０１」を３進数の「１」、「１０」を３進数の「２」に対応付けると、２ビット単位の２進表現の乱数列から３進数の乱数列を得ることができる。但し、２ビットの値「１１」は除外する。つまり、抽出方法＃１は、２進数２シンボルで表現される2^2通りの数の中から、３進数１シンボルで表現される3^1通りの数を抽出する方法である。従って、Ｎ個以上の３進数が抽出できない確率P_1は、下記の式（３０）のようになる。
【０２１７】
【数１６】

$$P_1 = \sum_{M/2-N<i\leq M/2} {}_{M/2}C_i (1/4)^i (3/4)^{M/2-i}$$

…（３０）

【０２１８】
［６－２：抽出方法＃２（区切りなし）（図１８）］
次に、図１８を参照しながら、２進数Ｍシンボルの乱数を区切らずに利用して３進数Ｌシンボルの乱数を抽出する方法（以下、抽出方法＃２）を紹介する。但し、Ｌは、$3^L \leqq 2^M$となる最大の整数である。２進数Ｍシンボルで表現可能な数は、2^M通り存在する。一方、３進数Ｌシンボルで表現可能な数は、3^L通りしか存在しない。そのため、２進数Ｍシンボルで表現される2^M通りの数のうち、$2^M - 3^L$通りの数は、３進表現の乱数として利用されない。従って、Ｎ個以上の３進数が抽出できない確率P_2は、下記の式（３１）のようになる。
【０２１９】
【数１７】

$$P_2 = 1 - 3^L / 2^M$$

…（３１）

【０２２０】
［６－３：抽出方法＃３（ｋビット区切り）（図１９）］
上記の抽出方法＃１は、最小の区切り単位で２進表現の乱数列を区切る方法である。一方、上記の抽出方法＃２は、（Ｍビット区切りと考えられるため）最大の区切り単位で２進表現の乱数列を区切る方法である。上記の式（３０）及び式（３１）から分かるように

、区切る長さに応じて、Ｎ個以上の３進数が抽出できない確率は異なる。因みに、図１９に示すように、ｋビット単位で２進Ｍシンボルの乱数列を区切った場合、Ｎ個以上の３進数が抽出できない確率$Ｐ_{3}$は、下記の式（３２）のようになる。

【０２２１】

【数１８】

$$P_3 = \sum_{M/k-N/L<i\leq M/k} {}_{M/k}C_i (1-3^L/2^k)^i (3^L/2^k)^{M/k-i}$$

…（３２）

【０２２２】

Ｎ個以上の３進数が抽出できない確率$Ｐ_{3}$を最小化することができれば、最も効率よく３進表現の乱数列を抽出できることになる。例えば、Ｍ＝５１２、Ｎ＝１４０の場合、ｋ＝８のときに確率$Ｐ_{3}$が最小となる。

【０２２３】

（６－３－１：基本構成（図２０））

ここで、図２０を参照しながら、２進Ｍシンボルの乱数列から３進Ｌシンボルの乱数列を抽出する処理の流れについて説明する。図２０に示すように、まず、２進Ｍシンボルの乱数列が生成される（Ｓ２０１）。次いで、２進Ｍシンボルの乱数列が、ｋビット単位に区切られる（Ｓ２０２）。次いで、ｋビット単位で区切られたビット列$Ｘ_{2^k}$のうち、$Ｘ_{2^k} \leqq 3^{Ｌ}$となるビット列が抽出される（Ｓ２０３）。次いで、抽出されたビット列が、３進数表現で出力され（Ｓ２０４）、一連の処理が終了する。

【０２２４】

（６－３－２：追加抽出方法（図２１））

上記の式（１５）で表現される確率$Ｐ_{3}$が最小となる区切りの長さｋを算出し、図２０に示したアルゴリズムを実行することにより、２進表現の乱数列から３進表現の乱数列を効率的に抽出することが可能になる。しかし、本件発明者は、図２０のステップＳ２０４において$Ｘ_{2^k} > 3^{Ｌ}$となるビット列が利用されない点に注目し、３進表現の乱数列をより効率的に抽出する方法を考案した。以下、この方法について、図２１を参照しながら説明する。

【０２２５】

この方法は、図２０のステップＳ２０４で抽出されなかったビット列を利用して、３進表現のシンボル列を抽出する方法である。図２１に示すように、まず、図２０のステップＳ２０４で抽出されなかった、$Ｘ_{2^k} > 3^{Ｌ}$となるビット列の組（例えば、ビット列の組を$ｙ_{1}ｙ_{2}\cdots ｙ_{Ｎ}$，と表現すると、個々のビット列$ｙ_{ｉ}$は$3^{Ｌ} \leqq ｙ_{ｉ} < 2^{ｋ}$）が抽出される（Ｓ２１１）。次いで、抽出したビット列$ｙ_{ｉ}$からそれぞれ$3^{Ｌ}$が減算され、新たなビット列の組（例えば、新たなビット列の組を$ｚ_{1}ｚ_{2}\cdots ｚ_{Ｎ}$，と表現すると、個々のビット列$ｚ_{ｉ} = ｙ_{ｉ} - 3^{Ｌ}$は$0 \leqq ｚ_{ｉ} < 2^{ｋ} - 3^{Ｌ}$）が算出される（Ｓ２１２）。

【０２２６】

次いで、新たなビット列の組から、ビット列Ｘが$Ｘ < 3^{Ｌ'}$となるものが抽出される（Ｓ２１３）。但し、Ｌ’は、$3^{Ｌ'} \leqq 2^{ｋ} - 3^{Ｌ}$となる最大の整数である。次いで、ステップＳ２１３で抽出されたビット列が３進表現で出力され（Ｓ２１４）、一連の処理が終了する。このアルゴリズムを適用することにより、確率$3^{Ｌ'}／（2^{ｋ} - 3^{Ｌ}）$で新たにＬ’個の３進数を抽出することが可能になる。なお、この方法を再帰的に用いることで、さらに多くの３進数を抽出することができる。つまり、ステップＳ２１３でビット列Ｘが$Ｘ \geqq 3^{Ｌ'}$となったビット列から同様にして３進数を抽出することができる。

【０２２７】

以上、２進数の一様乱数から３進数の一様乱数を効率的に生成する方法について説明した。

【０２２８】
＜７：多変数多項式の係数を効率的に代入する方法について＞
さて、これまでは多変数多項式を証明者（又は署名者）と検証者との間で共有する方法について具体的に明示してこなかった。多変数多項式を共有する方法としては、多変数多項式の係数（乱数）を生成する際に用いるシードを両者で共有する方法が考えられる。しかしながら、共有しているシードを利用して生成した乱数を係数に当てはめる順序を両者で共有しない限り、多変数多項式を共有したことにはならない。
【０２２９】
［７－１：基本的な取り決め］
そこで、証明者（又は署名者）と検証者との間で、共有しているシードを用いて生成した乱数列を、どの順序で多変数多項式に当てはめるかについて基本的な取り決めを行う。そして、多変数多項式を利用する際に、この基本的な取り決めに従って乱数列を多変数多項式に当てはめる。このような方法を用いれば、証明者（又は署名者）と検証者との間で多変数多項式を共有したことになる。
【０２３０】
［７－２：データの構造化］
但し、多変数多項式を構成する係数の数は膨大である。１つの係数が１ビット単位で表現されている場合、多変数多項式を表現するのに最低でも数万ビット以上のデータが必要になる。そのため、数を多変数多項式の係数に代入する処理の負荷は非常に高い。そこで、本件発明者は、多変数多項式の係数を所定の単位で構造化し、係数への代入処理を効率化する手法（構造化手法＃１及び＃２）を考案した。さらに、本件発明者は、同じ多変数多項式の係数に対して複数回の代入処理を実行する場合に、その処理効率を向上させる手法（構造化手法＃３）を考案した。以下、これらの手法について詳細に説明する。
【０２３１】
（７－２－１：構造化手法＃１（図２２～図２６））
まず、構造化手法＃１について説明する。構造化手法＃１は、図２２及び図２３に示すように、多変数多項式を構成する同じ種類の項の係数を１つのデータ構造として纏める手法である。図２２の例では、多変数多項式Ｆの係数a_{1IJ}～a_{MIJ}がデータ構造Ａとして纏められ、係数b_{1I}～b_{MI}がデータ構造Ｂとして纏められている。また、図２３に示すように、多変数多項式Ｇについても同様の手法が適用可能である。この場合、係数（$a_{1IJ}+a_{1JI}$）～（$a_{MIJ}+a_{MJI}$）がデータ構造として纏められる。
【０２３２】
構造化手法＃１を適用しない場合、Ｍ本のＮ変数多項式で構成される多変数多項式Ｆの計算は、（例１）に示すようなアルゴリズムにより行われる。（例１）の場合、１ビットのＡＮＤ演算（＆）を２×Ｎ×（Ｎ＋１）×Ｍ／２回実行する必要がある。さらに、１ビットのＸＯＲ演算（＾）を１×Ｎ×（Ｎ＋１）×Ｍ／２回実行する必要がある。なお、入力ｘはＮビットであるとする。
【０２３３】
（例１）

```
input x;
for L=1 to M
  for I=1 to N
    for J=I to N
      [yのLビット目] ^= [a_LIJ] & [xのIビット目] & [xのJビット目];
    end for
  end for
end for
output y;
```

【０２３４】

一方、図２２に示すように係数を構造化し、生成した乱数を多変数多項式Ｆの係数として一部ずつ逐次適用していく場合、係数の代入アルゴリズムは（例２）のようになる。（例２）の場合、ＭビットのＡＮＤ演算（＆）を２×Ｎ×（Ｎ＋１）／２回、ＭビットのＸＯＲ演算（＾）をＮ×（Ｎ＋１）／２回実行するだけで済む。なお、$a_{(1\sim M)IJ}$は、それぞれループのタイミングで生成される。また、係数を使い回してもよい。例えば、Ｎ（Ｎ－１）／２回のループを実行する際、［$a_{(1\sim M)IJ}$］を毎回生成せず、Ｍ回に１回だけ生成するようにしてもよい。また、Ｍ回のループ中、［$a_{(1\sim M)IJ}$］を１ビットずつローテーションしながら利用してもよい。

【０２３５】

（例２）

```
input x;
for I=1 to N
  for J=I to N
    [yの1～Mビット目]^=[a(1～M)IJ]&[xのIビット目]&[xのJビット目];
  end for
end for
output y;
```

【０２３６】

また、図２２に示すように係数を構造化する場合、多変数多項式Ｆの係数を適用した中間の結果をテーブルに保持するようにしてもよい。この場合、上記のアルゴリズムは（例３）のようになる。なお、配列$a_{IJ}[0][0]\sim a_{IJ}[2^k-1][2^k-1]$には、それぞれ、$a_{IJ}[x_1,\cdots,x_k][z_1,\cdots,z_k]=(a_{(k(I-1)+1)(k(J-1)+1)}\&x_1\&z_1)\hat{}\cdots\hat{}(a_{(k(I-1)+1)(k(J-1)+k)}\&x_1\&z_k)\hat{}\cdots\hat{}(a_{(k(I-1)+k)(k(J-1)+1)}\&x_k\&z_1)\hat{}\cdots\hat{}(a_{(k(I-1)+k)(k(J-1)+k)}\&x_k\&z_k)$が格納されている。（例３）の場合、ＭビットのＸＯＲ演算（＾）を（Ｎ／ｋ）（Ｎ／ｋ＋１）／２回実行するだけで済む。但し、（例２）のアルゴリズムに比べると、必要なメモリ量は、$2^{2k}/k^2$倍となる。

【０２３７】

例えば、ｋ＝１のとき、ＭビットＸＯＲ演算が１２０＊１１９／２＝７１４０回、必要なメモリ量が（例２）の2^2＝４倍、ループ回数は変化なしとなる。また、ｋ＝２のとき、ＭビットＸＯＲ演算が６０＊５９／２＝１７７０回、必要なメモリ量が2^4／４＝４倍、ループ回数が１／４となる。ｋ＝４のとき、ＭビットＸＯＲ演算が３０＊２９／２＝４３５回、必要なメモリ量が2^8／4^2＝１６倍、ループ回数が１／１６．４となる。ｋ＝６のとき、ＭビットＸＯＲ演算が２０＊１９／２＝１９０回、必要なメモリ量が2^{12}／6^2＝１１４倍、ループ回数が１／３７．６となる。ｋ＝８のとき、ＭビットＸＯＲ演算が１５＊１４／２＝１３５回、必要なメモリが2^{16}／8^2＝１０２４倍、ループ回数が１／５２．９となる。

【０２３８】

（例３）

```
input x;
for I=1 to N/k
  for J=I to N/k
    [yの1～Mビット目]^=a(1～M)IJ[xのk（I－1）＋1～kビット目][xのk（J－1）＋1～kビット目];
  end for
end for
output y;
```

【０２３９】

なお、（例３）に示した手法は、ちょうど、下記の式（３３）で定義されるＦＩＪ（…）の値を予め計算し、配列として保持しておく手法と言える。

【０２４０】

【数１９】

$$F(x_1,\cdots,x_n)=\sum_{I,J}F_{IJ}(x_{k(I-1)+1},\cdots,x_{k(I-1)+k},x_{k(J-1)+1},\cdots,x_{k(J-1)+k})=y$$

…　（３３）

但し、$F_{IJ}(x_{k(I-1)+1},\cdots,x_{k(I-1)+k},x_{k(J-1)+1},\cdots,x_{k(J-1)+k})$は、$F(x_1,\cdots,x_N)$のうち、

$[x_{k(I-1)+1},\cdots,x_{k(I-1)+k}]$及び$[x_{k(J-1)+1},\cdots,x_{k(J-1)+k}]$により値が定まる部分を表す。

【０２４１】

以上、構造化手法＃１を多変数多項式Ｆへ適用するケースを例に挙げて具体的なアルゴリズムについて説明した。かかる構成により、アルゴリズムを実行する際に処理の高速化が期待できる。

【０２４２】

（Ｇへの適用）

ここまで、図２２を参照しながら、構造化手法＃１を適用して多変数多項式Ｆを計算するアルゴリズムについて説明してきた。一方で、多変数多項式Ｇの各要素も二次形式で表現されることから、図２３に示すように、同様にして上記の構造化手法＃１を多変数多項式Ｇの計算にそのまま適用することも可能である。

【０２４３】

例えば、上記の構造化手法＃１を適用しない場合、多変数多項式Ｇを計算するアルゴリズムは、下記の（例１’）のように表現される。なお、入力ｘ及びｙはそれぞれＮビットであるとする。

【０２４４】

（例１’）

```
input x, y;
for L=1 to M
  for I=1 to N
    for J=1 to N
      [zのLビット目] ^= [a_LIJ + a_LJI] & [xのIビット目] & [yのJビット目];
    end for
  end for
end for
output z;
```

【０２４５】

上記の（例１’）に対し、上述した構造化手法＃１を適用すると、多変数多項式Ｇを計算するアルゴリズムは、下記の（例２’）のようになる。

【０２４６】

（例２’）

```
input x, y;
for I=1 to N
  for J=1 to N
    [zの1～Mビット目] ^= [a_(1～M)IJ + a_(1～M)JI] & [xのIビット目] & [yのJビット目];
  end for
```

end for
output z;

【０２４７】

また、多変数多項式Ｇの係数を適用した中間の結果をテーブルに保持する手法の場合、多変数多項式Ｇを計算するアルゴリズムは、上記の（例３）に対応して下記の（例３’）のようになる。なお、配列$a_{IJ}[0][0]$～$a_{IJ}[2^k-1][2^k-1]$には、それぞれ、$a_{IJ}[x_1,\cdots,x_k][y_1,\cdots,y_k]=(a_{(k(I-1)+1)(k(J-1)+1)}\&x_1\&y_1)\wedge\cdots\wedge(a_{(k(I-1)+1)(k(J-1)+k)}\&x_1\&y_k)\wedge\cdots\wedge(a_{(k(I-1)+k)(k(J-1)+1)}\&x_k\&y_1)\wedge\cdots\wedge(a_{(k(I-1)+k)(k(J-1)+k)}\&x_k\&y_k)$が格納されている。

【０２４８】

（例３’）の場合、ＭビットのＸＯＲ演算（∧）を$(N/k)^2$回実行するだけで済む。但し、（例２’）のアルゴリズムに比べると、必要なメモリ量は、$2^{2k}/k^2$倍となる。

【０２４９】

例えば、ｋ＝１のとき、ＭビットＸＯＲ演算が$120^2=14400$回、必要なメモリ量が（例２’）の$2^2=4$倍、ループ回数は変化なしとなる。また、ｋ＝２のとき、ＭビットＸＯＲ演算が$60^2=3600$回、必要なメモリ量が$2^4/4=4$倍、ループ回数が１／４となる。ｋ＝４のとき、ＭビットＸＯＲ演算が$30^2=900$回、必要なメモリ量が$2^8/4^2=16$倍、ループ回数が１／１６となる。ｋ＝６のとき、ＭビットＸＯＲ演算が$20^2=400$回、必要なメモリ量が$2^{12}/6^2=114$倍、ループ回数が１／３６となる。ｋ＝８のとき、ＭビットＸＯＲ演算が$15^2=225$回、必要なメモリが$2^{16}/8^2=1024$倍、ループ回数が１／６４となる。

【０２５０】

（例３’）
input x, y;
for I=1 to N／k
for J=I to N／k
［zの１～Mビット目］∧＝$a_{(1\sim M)IJ}$［xのｋ（Ｉ－１）＋１～ｋビット目］［yのｋ（Ｊ－１）＋１～ｋビット目］;
end for
end for
output z;

【０２５１】

なお、（例３’）に示した手法は、ちょうど、下記の式（３４）で定義されるG_{IJ}（…）の値を予め計算し、配列として保持しておく手法と言える。

【０２５２】

【数２０】

$$G(x_1,\cdots,x_n)=\sum_{I,J}G_{IJ}(x_{k(I-1)+1},\cdots,x_{k(I-1)+k},y_{k(J-1)+1},\cdots,y_{k(J-1)+k})=z$$

…　（３４）

但し、$G_{IJ}(x_{k(I-1)+1},\cdots,x_{k(I-1)+k},y_{k(J-1)+1},\cdots,y_{k(J-1)+k})$は、$G(x_1,\cdots,x_N,y_1,\cdots,y_N)$のうち、

$[x_{k(I-1)+1},\cdots,x_{k(I-1)+k}]$及び$[y_{k(J-1)+1},\cdots,y_{k(J-1)+k}]$により値が定まる部分を表す。

【０２５３】

以上、多変数多項式Ｇを計算するケースを例に挙げ、構造化手法＃１に係る具体的なア

ルゴリズムの構成について説明した。かかる構成により、アルゴリズムを実行する際に処理の高速化が期待できる。

【０２５４】

（３次多変数多項式Ｆへの適用）

ここまで、構造化手法＃１の具体例として、２次の多変数多項式の係数を構造化する手法を紹介してきたが、同様にして３次の多変数多項式の係数を構造化することも可能である。図２４の例では、係数a_{1IJL}～a_{MIJL}がデータ構造Ａとして纏められ、係数b_{1IJ}～b_{MIJ}がデータ構造Ｂとして纏められ、係数c_{1I}～c_{MI}がデータ構造Ｃとして纏められる。このように係数を構造化することで多変数多項式の計算が効率化される。

【０２５５】

例えば、データ構造Ａに係る計算について具体的に考えてみたい。係数を構造化しない場合、当該データ構造Ａに係る計算のアルゴリズムは、例えば、下記（例４）のようになる。

【０２５６】

（例４）

```
input x;
for Q=I to M
  for I=1 to N
    for J=I to N
      for L=J to N
        [yのQビット目]^=[a_QIJL]&[xのＩビット目]&[xのＪビット目]&[xのＬビット目];
      end for
    end for
  end for
end for
output y;
```

【０２５７】

一方、係数を構造化すると、下記（例５）のようなアルゴリズムとなる。（例５）の場合、ＭビットのＡＮＤ演算（＆）を２×Ｎ×（Ｎ＋１）×（Ｎ＋２）／６回、ＭビットのＸＯＲ演算（＾）を１×Ｎ×（Ｎ＋１）×（Ｎ＋２）／６回実行するだけで済む。なお、$a_{(1～M)IJL}$は、それぞれループのタイミングで生成される。また、係数を使い回してもよい。例えば、ループを実行する際、［$a_{(1～M)IJL}$］を毎回生成せず、Ｍ回に１回だけ生成するようにしてもよい。また、Ｍ回のループ中、［$a_{(1～M)IJ}$］を１ビットずつローテーションしながら利用してもよい。

【０２５８】

（例５）

```
input x;
for I=1 to N
  for J=I to N
    for L=J to N
      [yの１～Ｍビット目]^=[a_(1～M)IJL]&[xのＩビット目]&[xのＪビット目]&[xのＬビット目];
    end for
  end for
end for
output y;
```

【０２５９】

また、係数を構造化する場合、多変数多項式Ｆの係数を適用した中間の結果をテーブルに保持するようにしてもよい。この場合、上記のアルゴリズムは（例６）のようになる。

なお、配列$a_{IJL}[0][0][0]$～$a_{IJL}[2^k-1][2^k-1][2^k-1]$には、それぞれ、$a_{IJL}[x_1,\cdots,x_k][z_1,\cdots,z_k][w_1,\cdots,w_k]=\Sigma_{1\leqq\alpha,\beta,\gamma\leqq k}(a_{(k(I-1)+\alpha)(k(J-1)+\beta)(k(L-1)+\gamma)}\&x_\alpha\&z_\beta\&w_\gamma)$が格納されている。（例６）の場合、ＭビットのＸＯＲ演算（＾）を（Ｎ／ｋ）（Ｎ／ｋ＋１）（Ｎ／ｋ＋２）／６回実行するだけで済む。但し、（例４）のアルゴリズムに比べると、必要なメモリ量は、$2^{3k}/k^3$倍となる。

【０２６０】

例えば、ｋ＝１のとき、ＭビットＸＯＲ演算が１２０＊１２１＊１２２／６＝２９５２４０回、必要なメモリ量が（例４）の$2^3=8$倍、ループ回数は変化なしとなる。また、ｋ＝２のとき、ＭビットＸＯＲ演算が６０＊６１＊６２／６＝３７８２０回、必要なメモリ量が$2^6/8=8$倍、ループ回数が１／７．８となる。ｋ＝４のとき、ＭビットＸＯＲ演算が３０＊３１＊３２／６＝４９６０回、必要なメモリ量が$2^{12}/4^3=64$倍、ループ回数が１／５９．５となる。ｋ＝６のとき、ＭビットＸＯＲ演算が２０＊２１＊２２／６＝１５４０回、必要なメモリ量が$2^{18}/6^3=1214$倍、ループ回数が１／１９２となる。

【０２６１】

（例６）

```
input x;
for I=1 to N/k
  for J=I to N/k
    for L=J to N/k
      [yの１～Mビット目]＾=a(1～M)IJL[xのk(I－1)＋1～kビット目][xのk(J－1)＋1～kビット目][xのk(L－1)＋1～kビット目];
    end for
  end for
end for
output y;
```

【０２６２】

なお、（例６）に示した手法は、ちょうど、下記の式（３５）で定義される$F_{IJ}(\cdots)$の値を予め計算し、配列として保持しておく手法と言える。

【０２６３】

【数２１】

$$F(x_1,\cdots,x_n)=y$$
$$=\sum_{I,J}F_{IJ}(x_{k(I-1)+1},\cdots,x_{k(I-1)+k},x_{k(J-1)+1},\cdots,x_{k(J-1)+k},x_{k(L-1)+1},\cdots,x_{k(L-1)+k})$$

…　（３５）

但し、$F_{IJ}(x_{k(I-1)+1},\cdots,x_{k(I-1)+k},x_{k(J-1)+1},\cdots,x_{k(J-1)+k},x_{k(L-1)+1},\cdots,x_{k(L-1)+k})$は、

$F(x_1,\cdots,x_N)$のうち、$[x_{k(I-1)+1},\cdots,x_{k(I-1)+k}]$、$[x_{k(J-1)+1},\cdots,x_{k(J-1)+k}]$、

及び$[x_{k(L-1)+1},\cdots,x_{k(L-1)+k}]$により値が定まる部分を表す。

【０２６４】

以上、構造化手法＃１を３次の多変数多項式Ｆへ適用するケースについて説明した。かかる構成により、アルゴリズムを実行する際に処理の高速化が期待できる。

【０２６５】

（３次多変数多項式G_x，G_yへの適用）
上記の（例５）や（例６）と同様、図２５及び図２６に示すように、３次の多変数多項式G_x，G_yに対しても構造化手法＃１を適用することが可能である。構造化手法＃１を適用しない場合、３次の多変数多項式G_xのデータ構造Ａに係る計算は、例えば、下記の（例４’Ｘ）のアルゴリズムにより実行される。

【０２６６】

（例４’Ｘ）

```
input x, y;
for Q=1 to M
  for I=1 to N
    for J=1 to N
      for L=1 to N
        [zのQビット目] ^= [a_QIJL+a_QILJ+a_QLJI] & [yのIビット目] & [yのJビット目] & [xのLビット目];
      end for
    end for
  end for
end for
output z;
```

【０２６７】

上記の（例４’Ｘ）に対し、上述した構造化手法＃１を適用すると、多変数多項式G_xを計算するアルゴリズムは、下記の（例５’Ｘ）のようになる。また、上記の（例６）と同様にして多変数多項式G_xの係数を適用した中間の結果をテーブルに保持するようにしてもよい。

【０２６８】

（例５’Ｘ）

```
input x, y;
for I=1 to N
  for J=1 to N
    for L=1 to N
      [zの1～Mビット目] ^= [a_(1～M)IJL+a_(1～M)ILJ+a_(1～M)LJI] & [yのIビット目] & [yのJビット目] & [xのLビット目];
    end for
  end for
end for
output z;
```

【０２６９】

また、構造化手法＃１を適用しない場合、３次の多変数多項式G_yの計算は、例えば、下記の（例４’Ｙ）のアルゴリズムにより実行される。

【０２７０】

（例４’Ｙ）

```
input x, y;
for Q=1 to M
  for I=1 to N
    for J=1 to N
      for L=1 to N
        [zのQビット目] ^= [a_QIJL+a_QILJ+a_QLJI] & [xのIビット目] & [xのJビット目] & [yのLビット目];
      end for
    end for
```

```
    end for
  end for
  output z;
```

【０２７１】

上記の（例４’Ｙ）に対し、上述した構造化手法＃１を適用すると、多変数多項式G_yを計算するアルゴリズムは、下記の（例５’Ｙ）のようになる。また、上記の（例６）と同様にして多変数多項式G_yの係数を適用した中間の結果をテーブルに保持するようにしてもよい。

【０２７２】

（例５’Ｙ）

```
  input x, y;
  for I=1 to N
    for J=1 to N
      for L=1 to N
        [ｚの１～Mビット目]^=[a(1～M)IJL+a(1～M)ILJ+a(1～M)LJI]&[ｘのＩビット目]&[ｘのＪビット目]&[ｙのＬビット目];
      end for
    end for
  end for
  output z;
```

【０２７３】

以上、構造化手法＃１を３次の多変数多項式G_x、G_yへ適用するケースについて説明した。かかる構成により、アルゴリズムを実行する際に処理の高速化が期待できる。

【０２７４】

（７－２－２：構造化手法＃２（図２７））

次に、構造化手法＃２について説明する。構造化手法＃２は、図２７に示すように、多変数多項式を２次形式で表現し、２次形式の行又は列を１つのデータ構造として纏める手法である。図２７の例では、行方向にデータ構造が纏められている。

【０２７５】

図２７に示すように係数を構造化し、生成した乱数を多変数多項式の係数として一部ずつ逐次適用していく場合、係数の代入アルゴリズムは（例４）のようになる。例４の場合、ＮビットのＡＮＤ演算（&）を（Ｎ＋１）×M回、ＮビットのＸＯＲ演算（^）をＮ×M回、関数Ｑの演算をM回実行するだけで済む。

【０２７６】

（例４）

```
  for I=1 to N
    T^=A_I&[ｘのＩビット目];
  end for
  T&=x;
  output Q(T);
```

【０２７７】

```
  Q(z){
    z=z^(z>>1);
    z=z^(z>>2);
    z=z^(z>>4);
    z=z^(z>>8);
    ・・・
    z=z^(z>>2^Log(N));
    return z&1;
  }
```

【０２７８】
また、図２７に示すように係数を構造化し、多変数多項式の係数を適用した中間の結果をテーブルに保持するようにしてもよい。この場合、係数の代入アルゴリズムは（例５）のようになる。なお、配列$A_I[0]$～$A_I[2^k-1]$には、それぞれ、$A_I[x_1, \cdots, x_k] = (A_{(k(I-1)+1)} \& x_1) \wedge \cdots \wedge (A_{(k(I-1)+k)} \& x_k)$が格納されている。例５の場合、ＮビットのＸＯＲ演算（＾）を（Ｎ／ｋ）×Ｍ、Ｎビットの関数Ｑの演算をＭ回実行するだけで済む。但し、（例４）のアルゴリズムに比べると、必要なメモリ量は、2^k／ｋ倍となる。
【０２７９】
例えば、ｋ＝１のとき、ＮビットＸＯＲ演算が１２０回、必要なメモリ量が（例４）の２倍、ループ回数は変化なしとなる。また、ｋ＝４のとき、ＮビットＸＯＲ演算が３０回、必要なメモリ量が2^4／４＝４倍、ループ回数が１／４となる。ｋ＝８のとき、ＮビットＸＯＲ演算が１５回、必要なメモリが2^8／８＝３２倍、ループ回数が１／８となる。ｋ＝１６のとき、ＮビットＸＯＲ演算が８回、必要なメモリが2^{16}／１６＝４０９６倍、ループ回数が１／１５となる。
【０２８０】
（例５）
ｆｏｒ　Ｉ＝１　ｔｏ　Ｎ／ｋ
　Ｔ＾＝A_I［ｘのｋ（Ｉ－１）＋１～ｋ（Ｉ－１）＋ｋビット目］；
ｅｎｄ　ｆｏｒ
Ｔ＆＝ｘ；
ｏｕｔｐｕｔ　Ｑ（Ｔ）；
【０２８１】
以上、構造化手法＃２に係る具体的な係数の代入アルゴリズムについて説明した。かかる構成により、アルゴリズムを実行する際に処理の高速化が期待できる。
【０２８２】
（７－２－３：構造化手法＃３）
次に、構造化手法＃３について説明する。構造化手法＃３は、同じ多変数多項式にＮ回（Ｎ≧２）代入処理を行なう場合に、乱数から多項式をＮ回生成して代入処理を行うのではなく、「係数を一部生成し、その部分に関するＮ回分の処理を行なう」部分を単位とする逐次的な処理をＮ回分並列して行なうという手法である。この手法を適用すると、乱数生成のコストが無視できない場合にＮ回分の処理全体でのスループットが改善される。
【０２８３】
例えば、図５に示したアルゴリズムでは、工程＃１において引数を更新しながら多変数多項式Ｆ及びＧの計算をＮ回繰り返し実行している。そこで、このような計算部分について、同じ係数を利用して繰り返し演算を行うように構成する。上記（例２）のアルゴリズムを用いて多変数多項式Ｆ（r_{0i}）（ｉ＝１～Ｎ）を計算する場合、一度生成した［a_{IJL}］について、Ｎ個のr_{0i}を全て適用した後で、次の［a_{IJL}］に関する処理を実行するように構成する。このように構成すると、同じ係数［a_{IJL}］をＮ回生成せずに済むようになる。
【０２８４】
以上、構造化手法＃３に係る具体的な係数の代入アルゴリズムについて説明した。かかる構成により、Ｎ回分の処理におけるトータルでのスループットが改善される。
【０２８５】
＜８：ハードウェア構成例（図２８）＞
上記の各アルゴリズムは、例えば、図２８に示す情報処理装置のハードウェア構成を用いて実行することが可能である。つまり、当該各アルゴリズムの処理は、コンピュータプログラムを用いて図２８に示すハードウェアを制御することにより実現される。なお、このハードウェアの形態は任意であり、例えば、パーソナルコンピュータ、携帯電話、ＰＨＳ、ＰＤＡ等の携帯情報端末、ゲーム機、接触式又は非接触式のＩＣチップ、接触式又は

非接触式のＩＣカード、又は種々の情報家電がこれに含まれる。但し、上記のＰＨＳは、Ｐｅｒｓｏｎａｌ Ｈａｎｄｙ－ｐｈｏｎｅ Ｓｙｓｔｅｍの略である。また、上記のＰＤＡは、Ｐｅｒｓｏｎａｌ Ｄｉｇｉｔａｌ Ａｓｓｉｓｔａｎｔの略である。

【０２８６】

図２８に示すように、このハードウェアは、主に、ＣＰＵ９０２と、ＲＯＭ９０４と、ＲＡＭ９０６と、ホストバス９０８と、ブリッジ９１０と、を有する。さらに、このハードウェアは、外部バス９１２と、インターフェース９１４と、入力部９１６と、出力部９１８と、記憶部９２０と、ドライブ９２２と、接続ポート９２４と、通信部９２６と、を有する。但し、上記のＣＰＵは、Ｃｅｎｔｒａｌ Ｐｒｏｃｅｓｓｉｎｇ Ｕｎｉｔの略である。また、上記のＲＯＭは、Ｒｅａｄ Ｏｎｌｙ Ｍｅｍｏｒｙの略である。そして、上記のＲＡＭは、Ｒａｎｄｏｍ Ａｃｃｅｓｓ Ｍｅｍｏｒｙの略である。

【０２８７】

ＣＰＵ９０２は、例えば、演算処理装置又は制御装置として機能し、ＲＯＭ９０４、ＲＡＭ９０６、記憶部９２０、又はリムーバブル記録媒体９２８に記録された各種プログラムに基づいて各構成要素の動作全般又はその一部を制御する。ＲＯＭ９０４は、ＣＰＵ９０２に読み込まれるプログラムや演算に用いるデータ等を格納する手段である。ＲＡＭ９０６には、例えば、ＣＰＵ９０２に読み込まれるプログラムや、そのプログラムを実行する際に適宜変化する各種パラメータ等が一時的又は永続的に格納される。

【０２８８】

これらの構成要素は、例えば、高速なデータ伝送が可能なホストバス９０８を介して相互に接続される。一方、ホストバス９０８は、例えば、ブリッジ９１０を介して比較的データ伝送速度が低速な外部バス９１２に接続される。また、入力部９１６としては、例えば、マウス、キーボード、タッチパネル、ボタン、スイッチ、及びレバー等が用いられる。さらに、入力部９１６としては、赤外線やその他の電波を利用して制御信号を送信することが可能なリモートコントローラ（以下、リモコン）が用いられることもある。

【０２８９】

出力部９１８としては、例えば、ＣＲＴ、ＬＣＤ、ＰＤＰ、又はＥＬＤ等のディスプレイ装置、スピーカ、ヘッドホン等のオーディオ出力装置、プリンタ、携帯電話、又はファクシミリ等、取得した情報を利用者に対して視覚的又は聴覚的に通知することが可能な装置である。但し、上記のＣＲＴは、Ｃａｔｈｏｄｅ Ｒａｙ Ｔｕｂｅの略である。また、上記のＬＣＤは、Ｌｉｑｕｉｄ Ｃｒｙｓｔａｌ Ｄｉｓｐｌａｙの略である。そして、上記のＰＤＰは、Ｐｌａｓｍａ ＤｉｓｐｌａｙＰａｎｅｌの略である。さらに、上記のＥＬＤは、Ｅｌｅｃｔｒｏ－Ｌｕｍｉｎｅｓｃｅｎｃｅ Ｄｉｓｐｌａｙの略である。

【０２９０】

記憶部９２０は、各種のデータを格納するための装置である。記憶部９２０としては、例えば、ハードディスクドライブ（ＨＤＤ）等の磁気記憶デバイス、半導体記憶デバイス、光記憶デバイス、又は光磁気記憶デバイス等が用いられる。但し、上記のＨＤＤは、Ｈａｒｄ Ｄｉｓｋ Ｄｒｉｖｅの略である。

【０２９１】

ドライブ９２２は、例えば、磁気ディスク、光ディスク、光磁気ディスク、又は半導体メモリ等のリムーバブル記録媒体９２８に記録された情報を読み出し、又はリムーバブル記録媒体９２８に情報を書き込む装置である。リムーバブル記録媒体９２８は、例えば、ＤＶＤメディア、Ｂｌｕ－ｒａｙメディア、ＨＤ ＤＶＤメディア、各種の半導体記憶メディア等である。もちろん、リムーバブル記録媒体９２８は、例えば、非接触型ＩＣチップを搭載したＩＣカード、又は電子機器等であってもよい。但し、上記のＩＣは、Ｉｎｔｅｇｒａｔｅｄ Ｃｉｒｃｕｉｔの略である。

【０２９２】

接続ポート９２４は、例えば、ＵＳＢポート、ＩＥＥＥ１３９４ポート、ＳＣＳＩ、ＲＳ－２３２Ｃポート、又は光オーディオ端子等のような外部接続機器９３０を接続するためのポートである。外部接続機器９３０は、例えば、プリンタ、携帯音楽プレーヤ、デジ

タルカメラ、デジタルビデオカメラ、又はＩＣレコーダ等である。但し、上記のＵＳＢは、Ｕｎｉｖｅｒｓａｌ　Ｓｅｒｉａｌ　Ｂｕｓの略である。また、上記のＳＣＳＩは、Ｓｍａｌｌ　Ｃｏｍｐｕｔｅｒ　Ｓｙｓｔｅｍ　Ｉｎｔｅｒｆａｃｅの略である。

【０２９３】

通信部９２６は、ネットワーク９３２に接続するための通信デバイスであり、例えば、有線又は無線ＬＡＮ、Ｂｌｕｅｔｏｏｔｈ（登録商標）、又はＷＵＳＢ用の通信カード、光通信用のルータ、ＡＤＳＬ用のルータ、又は接触又は非接触通信用のデバイス等である。また、通信部９２６に接続されるネットワーク９３２は、有線又は無線により接続されたネットワークにより構成され、例えば、インターネット、家庭内ＬＡＮ、赤外線通信、可視光通信、放送、又は衛星通信等である。但し、上記のＬＡＮは、Ｌｏｃａｌ　Ａｒｅａ　Ｎｅｔｗｏｒｋの略である。また、上記のＷＵＳＢは、Ｗｉｒｅｌｅｓｓ　ＵＳＢの略である。そして、上記のＡＤＳＬは、Ａｓｙｍｍｅｔｒｉｃ　Ｄｉｇｉｔａｌ　Ｓｕｂｓｃｒｉｂｅｒ　Ｌｉｎｅの略である。

【０２９４】

＜９：まとめ＞

最後に、本技術の実施形態に係る技術内容について簡単に纏める。ここで述べる技術内容は、例えば、ＰＣ、携帯電話、ゲーム機、情報端末、情報家電、カーナビゲーションシステム等、種々の情報処理装置に対して適用することができる。なお、以下で述べる情報処理装置の機能は、１台の情報処理装置を利用して実現することも可能であるし、複数台の情報処理装置を利用して実現することも可能である。また、以下で述べる情報処理装置が処理を実行する際に用いるデータ記憶手段及び演算処理手段は、当該情報処理装置に設けられたものであってもよいし、ネットワークを介して接続された機器に設けられたものであってもよい。

【０２９５】

上記の情報処理装置の機能構成は以下のように表現される。例えば、下記（１）に記載の情報処理装置は、多次多変数連立方程式の求解困難性に安全性の根拠を置く効率的な公開鍵認証方式又は電子署名方式のアルゴリズムを実行する機能を有する。

【０２９６】

（１）

多次多変数多項式の組Ｆ＝（ｆ１，…，ｆｍ）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成部と、

前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、

を備え、

前記割当部は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、

情報処理装置。

【０２９７】

（２）

多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成部と、

前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、

を備え、

前記割当部は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次

形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り当て処理をグループ単位で実行する、
情報処理装置。
【０２９８】
（３）
前記数生成部は、前記割当部が１つのグループに対する割り当て処理を実行する際に、当該１つのグループに属する係数の分だけ前記数を生成する、
上記（１）又は（２）に記載の情報処理装置。
【０２９９】
（４）
前記数生成部は、前記１つのグループに属する係数の分だけ前記数を生成する際、１つの数を生成し、生成した数を１ビットずつローテーションして所要数の数を生成する、
上記（３）に記載の情報処理装置。
【０３００】
（５）
前記各グループに対応する種類の項に前記係数を割り当て、当該項の変数に任意の数を代入した値をテーブルとして保持するテーブル保持部をさらに備える、
上記（１）又は（２）に記載の情報処理装置。
【０３０１】
（６）
多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて前記多次多変数多項式の組Ｆを構成する各項の係数と同じ数だけ数を生成する数生成部と、
前記証明者と前記認証者との間で予め決められた順序で前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当部と、
を備える、
情報処理装置。
【０３０２】
（７）
前記情報は、乱数のシードであり、
前記所定の関数は、前記シードを利用して乱数を生成する乱数生成器である、
上記（１）～（６）のいずれか１項に記載の情報処理装置。
【０３０３】
（８）
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）、及びベクトル$s \in K^n$に基づいてメッセージを生成するメッセージ生成部と、
前記多次多変数多項式の組Ｆ及びベクトル$y=(y_1,\cdots,y_m)=(f_1(s),\cdots,f_m(s))$を保持する検証者に前記メッセージを提供するメッセージ提供部と、
ｋ通り（$k \geqq 3$）の検証パターンの中から前記検証者が選択した検証パターンに対応する回答情報を前記検証者に提供する回答提供部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記回答情報は、前記乱数の組及び前記メッセージの中から前記検証パターンに応じて選択される情報であり、
前記メッセージは、前記公開鍵及び前記回答情報を利用して、当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
上記（１）～（７）のいずれか１項に記載の情報処理装置。

【０３０４】
（９）
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）及びベクトルｙ＝（y_1，…，y_m）＝（f_1（ｓ），…，f_m（ｓ））を保持する情報保持部と、
前記多次多変数多項式の組Ｆ及びベクトルｓ∈K^nに基づいて生成されたメッセージを取得するメッセージ取得部と、
前記メッセージを提供した証明者に対し、ｋ通り（ｋ≧３）の検証パターンの中からランダムに選択した１つの検証パターンの情報を提供するパターン情報提供部と、
前記選択した検証パターンに対応する回答情報を前記証明者から取得する回答取得部と、
前記メッセージ、前記多次多変数多項式の組Ｆ、前記ベクトルｙ、及び前記回答情報に基づいて前記証明者が前記ベクトルｓを保持しているか否かを検証する検証部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記メッセージは、前記公開鍵及び前記回答情報を利用して、当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
上記（１）～（７）のいずれか１項に記載の情報処理装置。
【０３０５】
（１０）
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）、及びベクトルｓ∈K^nに基づいてメッセージを生成するメッセージ生成部と、
前記多次多変数多項式の組Ｆ及びベクトルｙ＝（y_1，…，y_m）＝（f_1（ｓ），…，f_m（ｓ））を保持する検証者に前記メッセージを提供するメッセージ提供部と、
前記検証者がランダムに選択した第１の情報及び前記メッセージを生成する際に得られる第２の情報を用いて第３の情報を生成する中間情報生成部と、
前記検証者に前記第３の情報を提供する中間情報提供部と、
ｋ通り（ｋ≧２）の検証パターンの中から前記検証者が選択した検証パターンに対応する回答情報を前記検証者に提供する回答提供部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記回答情報は、前記メッセージの中から前記検証パターンに応じて選択される情報であり、
前記メッセージは、前記公開鍵、前記第１の情報、前記第３の情報、前記回答情報を利用して、当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
上記（１）～（７）のいずれか１項に記載の情報処理装置。
【０３０６】
（１１）
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）及びベクトルｙ＝（y_1，…，y_m）＝（f_1（ｓ），…，f_m（ｓ））を保持する情報保持部と、
前記多次多変数多項式の組Ｆ及びベクトルｓ∈K^nに基づいて生成されたメッセージを取得するメッセージ取得部と、
前記メッセージを提供した証明者に対し、ランダムに選択した第１の情報を提供する情報提供部と、
前記第１の情報及び前記メッセージを生成する際に得られる第２の情報を用いて前記証明者が生成した第３の情報を取得する中間情報取得部と、
ｋ通り（ｋ≧３）の検証パターンの中からランダムに選択した１つの検証パターンの情報を前記証明者に提供するパターン情報提供部と、

前記選択した検証パターンに対応する回答情報を前記証明者から取得する回答取得部と、
前記メッセージ、前記第１の情報、前記第３の情報、前記多次多変数多項式の組Ｆ、及び前記回答情報に基づいて前記証明者が前記ベクトルｓを保持しているか否かを検証する検証部と、
を備え、
前記ベクトルｓは秘密鍵であり、
前記多次多変数多項式の組Ｆ及び前記ベクトルｙは公開鍵であり、
前記メッセージは、前記公開鍵、前記第１の情報、前記第３の情報、前記回答情報を利用して当該回答情報に対応する検証パターン用に予め用意された演算を実行することで得られる情報である、
上記（１）～（７）のいずれか１項に記載の情報処理装置。
【０３０７】
（１２）
前記アルゴリズムが複数回繰り返し実行される場合において、前記数生成部は１回だけ前記数を生成し、前記割当部は１回だけ前記割り当て処理を実行し、
前記アルゴリズムは、前記割当部が割り当てた係数を繰り返し利用する、
上記（１）又は（２）に記載の情報処理装置。
【０３０８】
（１３）
環Ｋ上で定義される多次多変数多項式の組Ｆ＝（f_1，…，f_m）、及び署名鍵$s \in K^n$を用いて文書Ｍに対する電子署名を生成する署名生成部と、
前記多次多変数多項式の組Ｆ及びベクトル$y=(f_1(s), \cdots, f_m(s))$を保持する検証者へと前記電子署名を提供する署名提供部と、
を備える、
上記（１）～（７）のいずれか１項に記載の情報処理装置。
【０３０９】
（１４）
多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成するステップと、
生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てるステップと、
を含み、
前記割り当てるステップでは、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、
情報処理方法。
【０３１０】
（１５）
多次多変数多項式の組Ｆ＝（f_1，…，f_m）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成するステップと、
生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てるステップと、
を含み、
前記割り当てるステップでは、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り

当て処理をグループ単位で実行する、
情報処理方法。
【０３１１】
（１６）
多次多変数多項式の組Ｆ＝（ｆ$_1$，…，ｆ$_m$）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて前記多次多変数多項式の組Ｆを構成する各項の係数と同じ数だけ数を生成するステップと、
前記証明者と前記認証者との間で予め決められた順序で前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てるステップと、
を含む、
情報処理方法。
【０３１２】
（１７）
多次多変数多項式の組Ｆ＝（ｆ$_1$，…，ｆ$_m$）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成機能と、
前記数生成機能が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当機能と、
をコンピュータに実現させるためのプログラムであり、
前記割当機能は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数のうち、変数の組み合わせの種類が同じ項の係数をグループ化しておき、割り当て処理をグループ単位で実行する、
プログラム。
【０３１３】
（１８）
多次多変数多項式の組Ｆ＝（ｆ$_1$，…，ｆ$_m$）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて、前記多次多変数多項式の組Ｆを構成する各項の係数に利用する数を生成する数生成機能と、
前記数生成機能が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当機能と、
をコンピュータに実現させるためのプログラムであり、
前記割当機能は、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式を２次形式で表現した場合の係数行列を行又は列毎にグループ化しておき、割り当て処理をグループ単位で実行する、
プログラム。
【０３１４】
（１９）
多次多変数多項式の組Ｆ＝（ｆ$_1$，…，ｆ$_m$）を含む公開鍵を利用した公開鍵認証方式又は電子署名方式のアルゴリズムを実行するエンティティの間で共有されている情報から所定の関数を用いて前記多次多変数多項式の組Ｆを構成する各項の係数と同じ数だけ数を生成する数生成機能と、
前記証明者と前記認証者との間で予め決められた順序で前記数生成部が生成した数を、前記多次多変数多項式の組Ｆを構成要素に含む多次多変数多項式の係数に割り当てる割当機能と、
をコンピュータに実現させるためのプログラム。
【０３１５】

（２０）
上記の（１９）に記載のプログラムが記録された、コンピュータにより読み取り可能な記録媒体。

【０３１６】
（備考）
上記の証明者アルゴリズムＰ、検証者アルゴリズムＶ、署名生成アルゴリズムＳｉｇ、署名検証アルゴリズムＶｅｒは、数生成部、割当部、テーブル保持部の一例である。上記の証明者アルゴリズムＰは、メッセージ生成部、メッセージ提供部、回答提供部、中間情報生成部、中間情報提供部の一例である。また、上記の検証者アルゴリズムＶは、情報保持部、メッセージ取得部、パターン情報提供部、回答取得部、検証部、中間情報取得部の一例である。

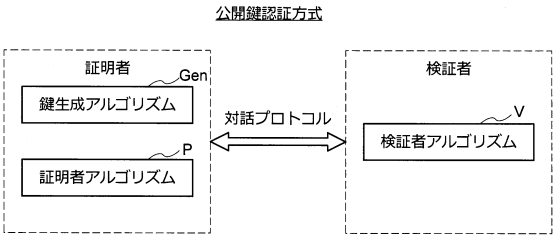
【０３１７】
以上、添付図面を参照しながら本技術の好適な実施形態について説明したが、本技術は係る例に限定されないことは言うまでもない。当業者であれば、特許請求の範囲に記載された範疇内において、各種の変更例または修正例に想到し得ることは明らかであり、それらについても当然に本技術の技術的範囲に属するものと了解される。

【０３１８】
上記の説明において、ハッシュ関数Ｈを用いるアルゴリズムを紹介したが、ハッシュ関数Ｈの代わりにコミットメント関数ＣＯＭを用いてもよい。コミットメント関数ＣＯＭは、文字列Ｓ及び乱数ρを引数にとる関数である。コミットメント関数の例としては、Ｓｈａｉ　ＨａｌｅｖｉとＳｉｌｖｉｏ　Ｍｉｃａｌｉによって国際会議ＣＲＹＰＴＯ１９９６で発表された方式などがある。

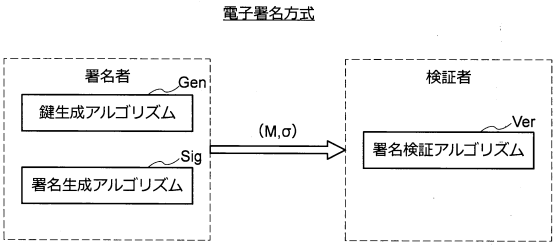
【符号の説明】

【０３１９】
Ｇｅｎ　鍵生成アルゴリズム
Ｐ　証明者アルゴリズム
Ｖ　検証者アルゴリズム
Ｓｉｇ　署名生成アルゴリズム
Ｖｅｒ　署名検証アルゴリズム

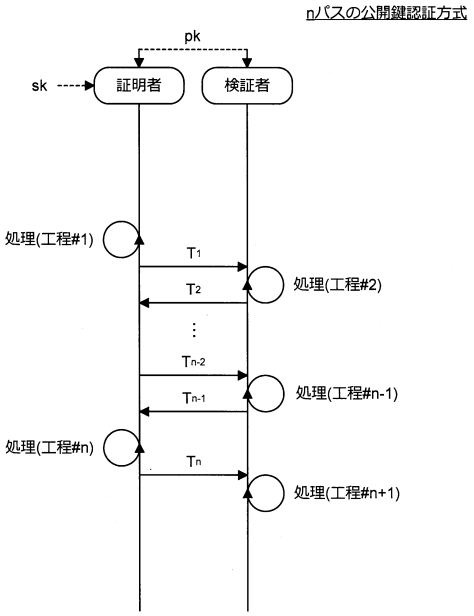
【図１】


公開鍵認証方式
証明者
Gen
鍵生成アルゴリズム
P
証明者アルゴリズム
対話プロトコル
検証者
V
検証者アルゴリズム

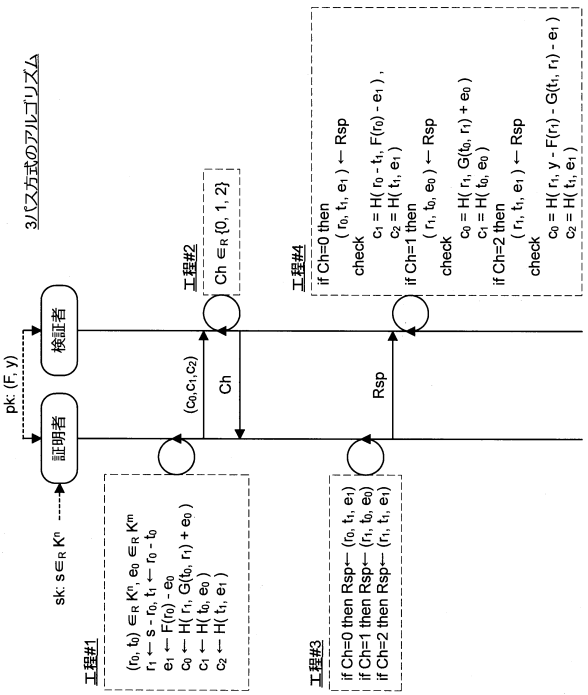
【図２】


電子署名方式
署名者
Gen
鍵生成アルゴリズム
Sig
署名生成アルゴリズム
(M,σ)
検証者
Ver
署名検証アルゴリズム

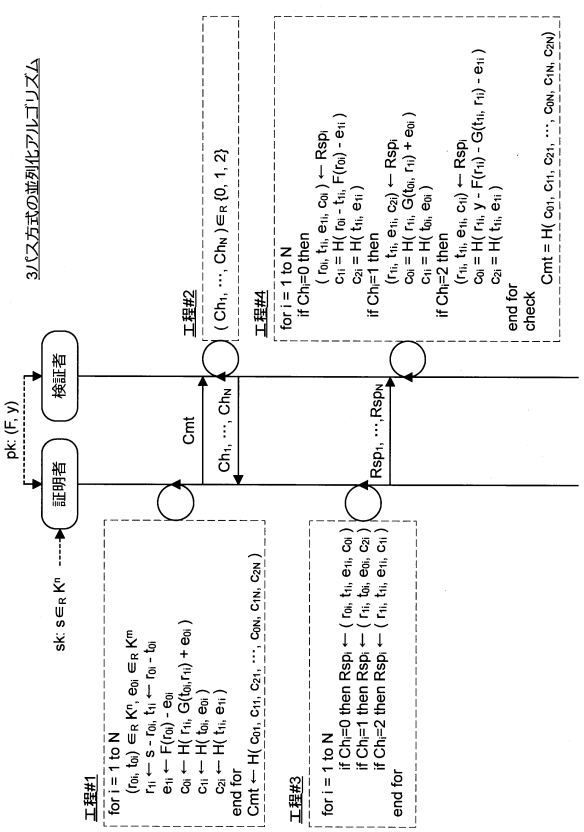
【図３】


nパスの公開鍵認証方式
pk
sk
証明者
検証者
処理(工程#1)
T1
処理(工程#2)
T2
Tn-2
Tn-1
処理(工程#n-1)
処理(工程#n)
Tn
処理(工程#n+1)

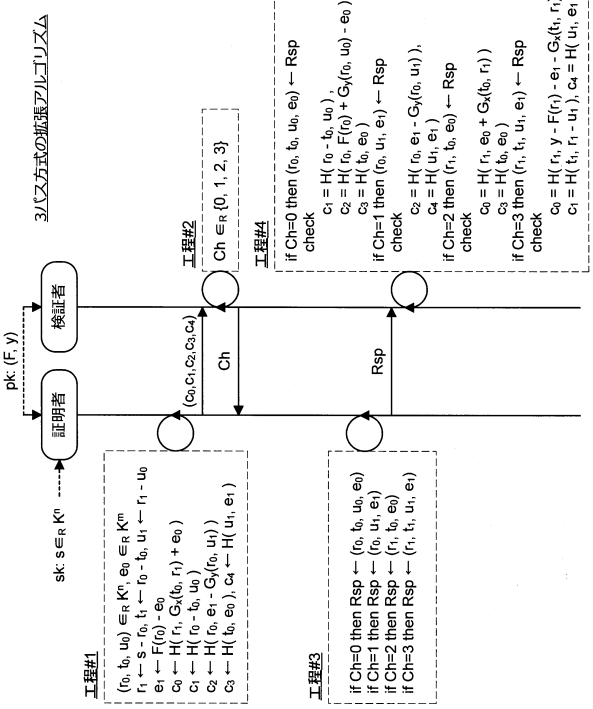
【図４】


3パス方式のアルゴリズム
pk: (F, y)
sk: s∈R K^n
検証者
証明者
(c0,c1,c2)
Ch
Rsp
工程#1
(r0, t0) ∈R K^n, e0 ∈R K^m
r1 ← s - r0, t1 ← r0 - t0
e1 ← F(r0) - e0
c0 ← H(r1, G(t0, r1) + e0)
c1 ← H(t0, e0)
c2 ← H(t1, e1)
工程#2
Ch ∈R {0, 1, 2}
工程#3
if Ch=0 then Rsp← (r0, t1, e1)
if Ch=1 then Rsp← (r1, t0, e0)
if Ch=2 then Rsp← (r1, t1, e1)
工程#4
if Ch=0 then
(r0, t1, e1) ← Rsp
check
c1 = H(r0 - t1, F(r0) - e1) ,
c2 = H(t1, e1)
if Ch=1 then
(r1, t0, e0) ← Rsp
check
c0 = H(r1, G(t0, r1) + e0)
c1 = H(t0, e0)
if Ch=2 then
(r1, t1, e1) ← Rsp
check
c0 = H(r1, y - F(r1) - G(t1, r1) - e1)
c2 = H(t1, e1)

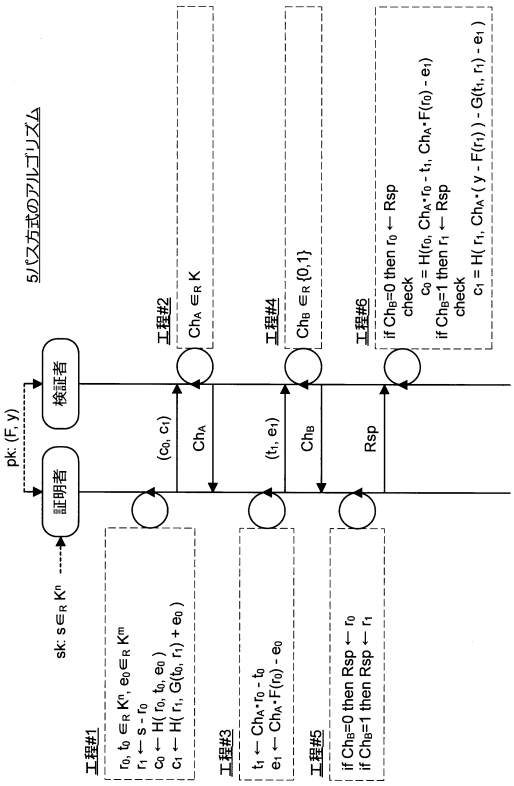
【図５】


3パス方式の並列化アルゴリズム
pk: (F, y)
sk: s∈R K^n
検証者
証明者
Cmt
Ch1, ..., ChN
Rsp1, ..., RspN
工程#1
for i = 1 to N
(r0i, t0i) ∈R K^n, e0i ∈R K^m
r1i ← s - r0i, t1i ← r0i - t0i
e1i ← F(r0i) - e0i
c0i ← H(r1i, G(t0i, r1i) + e0i)
c1i ← H(t0i, e0i)
c2i ← H(t1i, e1i)
end for
Cmt ← H(c01, c11, c21, ..., c0N, c1N, c2N)
工程#2
(Ch1, ..., ChN) ∈R {0, 1, 2}
工程#3
for i = 1 to N
if Chi=0 then Rspi ← (r0i, t1i, e1i, c0i)
if Chi=1 then Rspi ← (r1i, t0i, e0i, c2i)
if Chi=2 then Rspi ← (r1i, t1i, e1i, c1i)
end for
工程#4
for i = 1 to N
if Chi=0 then
(r0i, t1i, e1i, c0i) ← Rspi
c1i = H(r0i - t1i, F(r0i) - e1i)
c2i = H(t1i, e1i)
if Chi=1 then
(r1i, t0i, e0i, c2i) ← Rspi
c0i = H(r1i, G(t0i, r1i) + e0i)
c1i = H(t0i, e0i)
if Chi=2 then
(r1i, t1i, e1i, c1i) ← Rspi
c0i = H(r1i, y - F(r1i) - G(t1i, r1i) - e1i)
c2i = H(t1i, e1i)
end for
check
Cmt = H(c01, c11, c21, ..., c0N, c1N, c2N)

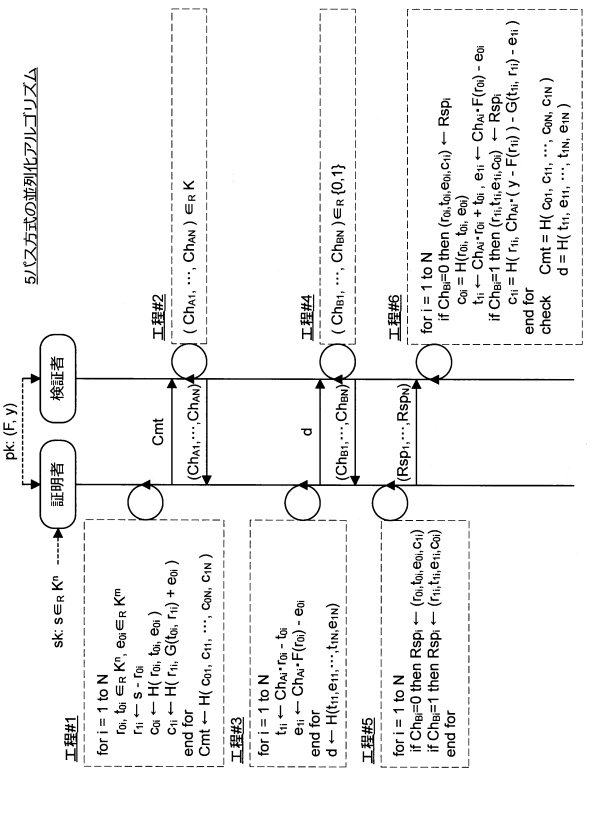
【図６】


3パス方式の拡張アルゴリズム
pk: (F, y)
sk: s∈R K^n
証明者
検証者
(c0,c1,c2,c3,c4)
Ch
Rsp
工程#1
(r0, t0, u0) ∈R K^n, e0 ∈R K^m
r1 ← s - r0, t1 ← r0 - t0, u1 ← r1 - u0
e1 ← F(r0) - e0
c0 ← H(r1, G_A(t0, r1) + e0)
c1 ← H(r0 - t0, u0)
c2 ← H(r0, e1 - G_y(r0, u1))
c3 ← H(t0, e0), c4 ← H(u1, e1)
工程#2
Ch ∈R {0, 1, 2, 3}
工程#3
if Ch=0 then Rsp ← (r0, t0, u0, e0)
if Ch=1 then Rsp ← (r0, u1, e1)
if Ch=2 then Rsp ← (r1, t0, e0)
if Ch=3 then Rsp ← (r1, t1, u1, e1)
工程#4
if Ch=0 then (r0, t0, u0, e0) ← Rsp
check
c1 = H(r0 - t0, u0),
c2 = H(r0, F(r0) + G_y(r0, u0) - e0),
c3 = H(t0, e0)
if Ch=1 then (r0, u1, e1) ← Rsp
check
c2 = H(r0, e1 - G_y(r0, u1)),
c4 = H(u1, e1)
if Ch=2 then (r1, t0, e0) ← Rsp
check
c0 = H(r1, e0 + G_A(t0, r1))
c3 = H(t0, e0)
if Ch=3 then (r1, t1, u1, e1) ← Rsp
check
c0 = H(r1, y - F(r1) - e1 - G_x(t1, r1))
c1 = H(t1, r1 - u1), c4 = H(u1, e1)

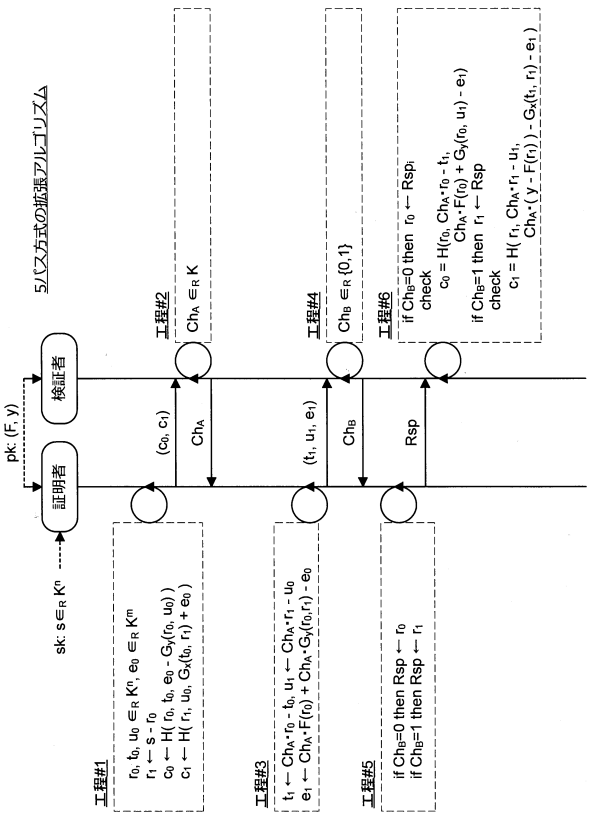
【図７】


5パス方式のアルゴリズム
pk: (F, y)
sk: s∈R K^n
証明者
検証者
(c0, c1)
Ch_A
(t1, e1)
Ch_B
Rsp
工程#1
r0, t0 ∈R K^n, e0∈R K^m
r1 ← s - r0
c0 ← H(r0, t0, e0)
c1 ← H(r1, G(t0, r1) + e0)
工程#2
Ch_A ∈R K
工程#3
t1 ← Ch_A・r0 - t0
e1 ← Ch_A・F(r0) - e0
工程#4
Ch_B ∈R {0,1}
工程#5
if Ch_B=0 then Rsp ← r0
if Ch_B=1 then Rsp ← r1
工程#6
if Ch_B=0 then r0 ← Rsp
check
c0 = H(r0, Ch_A・r0 - t1, Ch_A・F(r0) - e1
if Ch_B=1 then r1 ← Rsp
check
c1 = H(r1, Ch_A・(y - F(r1)) - G(t1, r1) - e1)

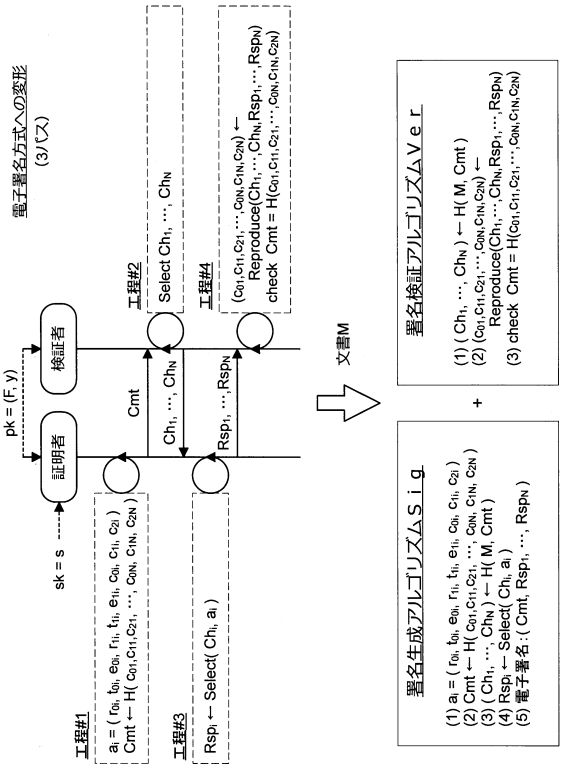
【図８】


5パス方式の並列化アルゴリズム
pk: (F, y)
sk: s∈R K^n
証明者
検証者
Cmt
(Ch_A1, ···, Ch_AN)
d
(Ch_B1, ···, Ch_BN)
(Rsp1, ···, RspN)
工程#1
for i = 1 to N
r0i, t0i ∈R K^n, e0i∈R K^m
r1i ← s - r0i
c0i ← H(r0i, t0i, e0i)
c1i ← H(r1i, G(t0i, r1i) + e0i)
end for
Cmt ← H(c01, c11, ···, c0N, c1N)
工程#2
(Ch_A1, ···, Ch_AN) ∈R K
工程#3
for i = 1 to N
t1i ← Ch_Ai・r0i - t0i
e1i ← Ch_Ai・F(r0i) - e0i
end for
d ← H(t11, e11, ···, t1N, e1N)
工程#4
(Ch_B1, ···, Ch_BN)∈R {0,1}
工程#5
for i = 1 to N
if Ch_Bi=0 then Rsp_i ← (r0i,t0i,e0i,c1i)
if Ch_Bi=1 then Rsp_i ← (r1i,t1i,e1i,c0i)
end for
工程#6
for i = 1 to N
if Ch_Bi=0 then (r0i,t0i,e0i,c1i) ← Rsp_i
c0i = H(r0i, t0i, e0i)
t1i ← Ch_Ai・r0i + t0i , e1i ← Ch_Ai・F(r0i) - e0i
if Ch_Bi=1 then (r1i,t1i,e1i,c0i) ← Rsp_i
c1i = H(r1i, Ch_Ai・(y - F(r1i)) - G(t1i, r1i) - e1i)
end for
Cmt = H(c01, c11, ···, c0N, c1N)
d = H(t11, e11, ···, t1N, e1N)

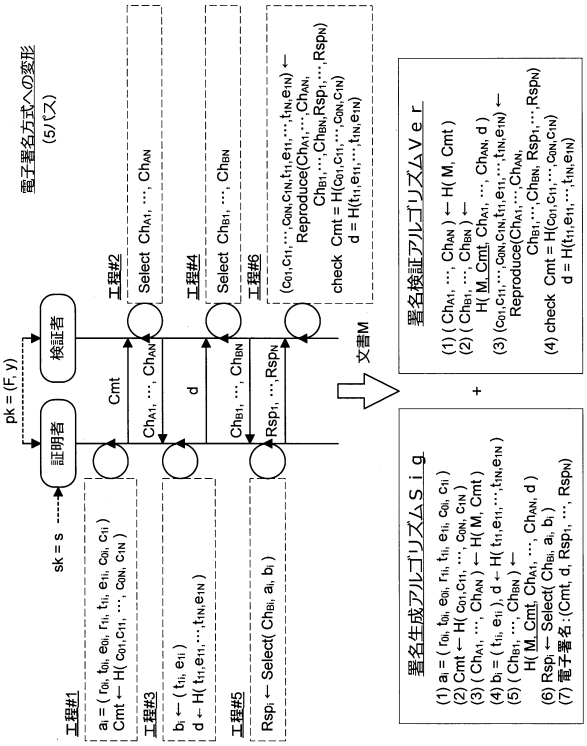
【図９】


5パス方式の拡張アルゴリズム
pk: (F, y)
sk: s∈R K^n
証明者
検証者
(c0, c1)
Ch_A
(t1, u1, e1)
Ch_B
Rsp
工程#1
r0, t0, u0 ∈R K^n, e0 ∈R K^m
r1 ← s - r0
c0 ← H(r0, t0, e0 - G_y(r0, u0))
c1 ← H(r1, u0, G_x(t0, r1) + e0)
工程#2
Ch_A ∈R K
工程#3
t1 ← Ch_A・r0 - t0, u1 ← Ch_A・r1 - u0
e1 ← Ch_A・F(r0) + Ch_A・G_y(r0,r1) - e0
工程#4
Ch_B ∈R {0,1}
工程#5
if Ch_B=0 then Rsp ← r0
if Ch_B=1 then Rsp ← r1
工程#6
if Ch_B=0 then r0 ← Rsp_i
check
c0 = H(r0, Ch_A・r0 - t1,
Ch_A・F(r0) + G_y(r0, u1) - e1)
if Ch_B=1 then r1 ← Rsp
check
c1 = H(r1, Ch_A・r1 - u1,
Ch_A・(y - F(r1)) - G_x(t1, r1) - e1)

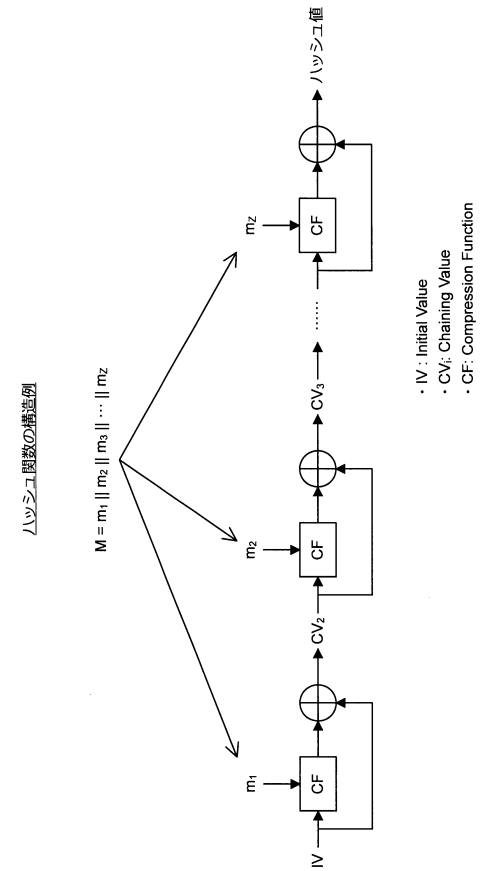
【図１０】



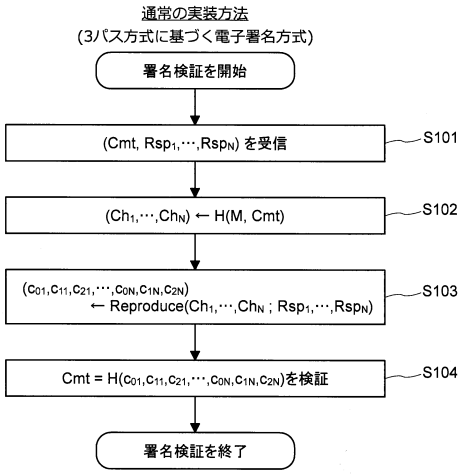
【図１１】



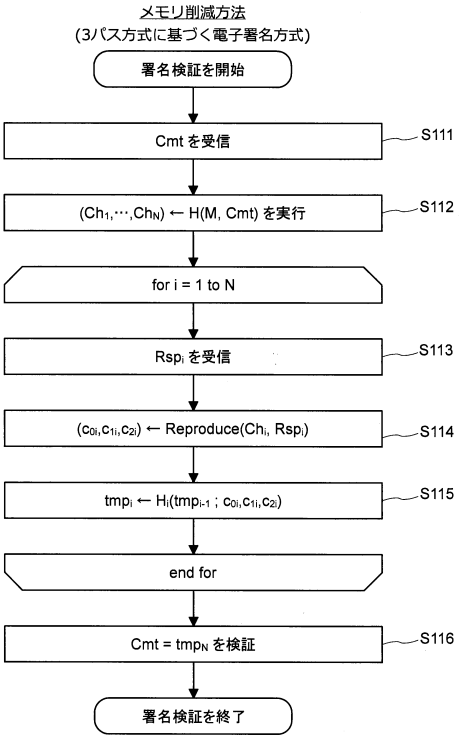
【図１２】



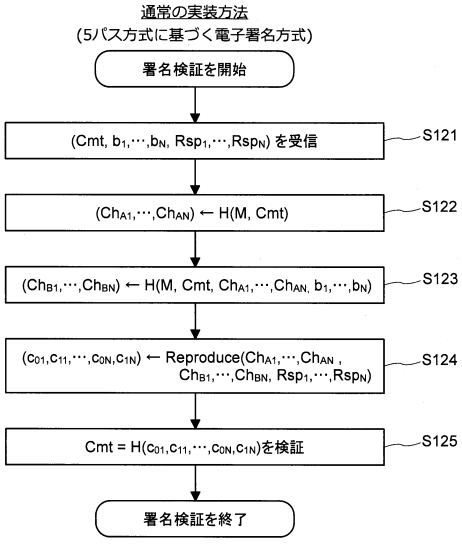
【図１３】



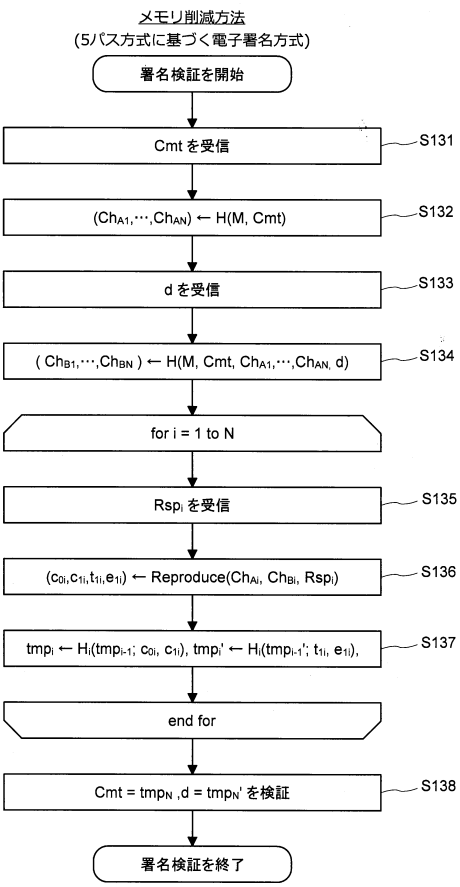
【図１４】



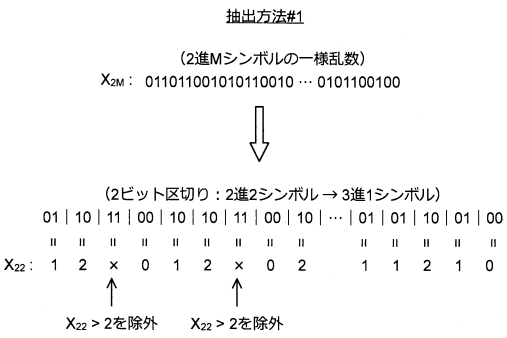
【図１５】



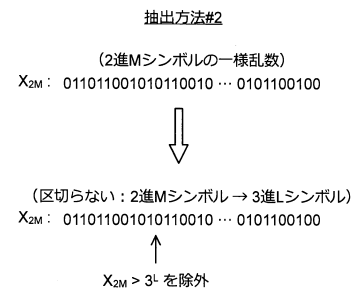
【図１６】



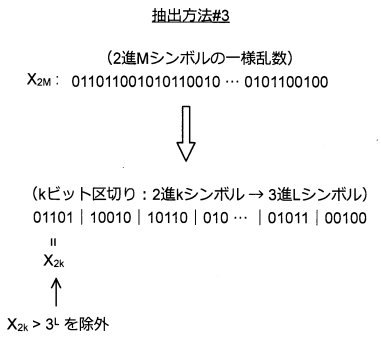
【図１７】



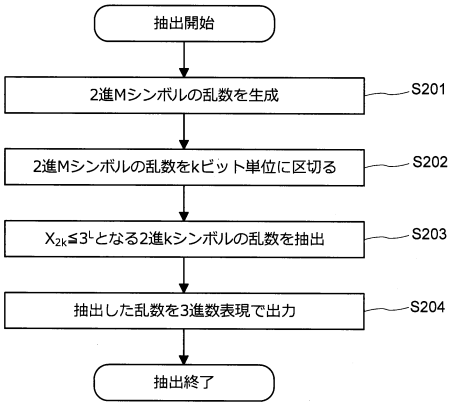
【図１８】



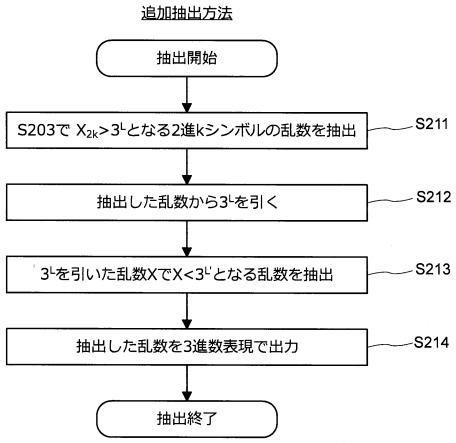
【図１９】



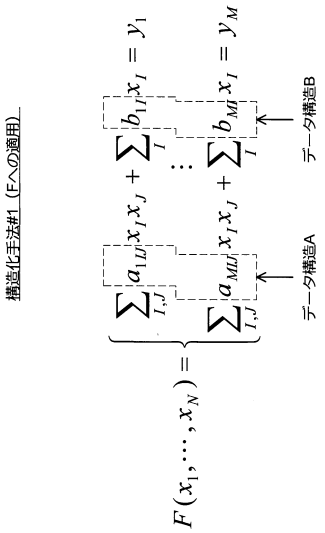
【図２０】



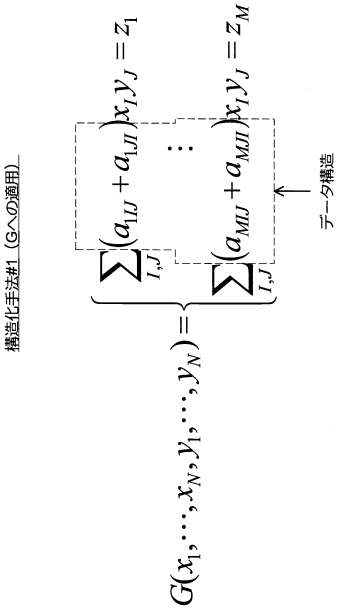
【図２１】



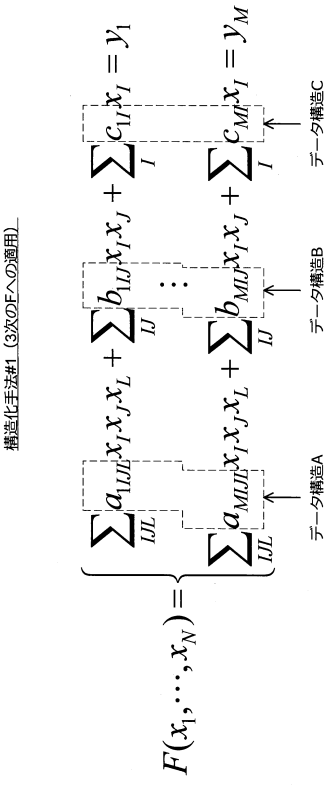
【図２２】



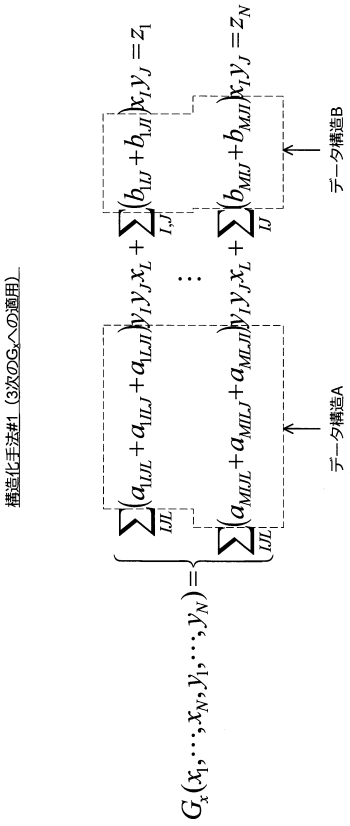
【図２３】



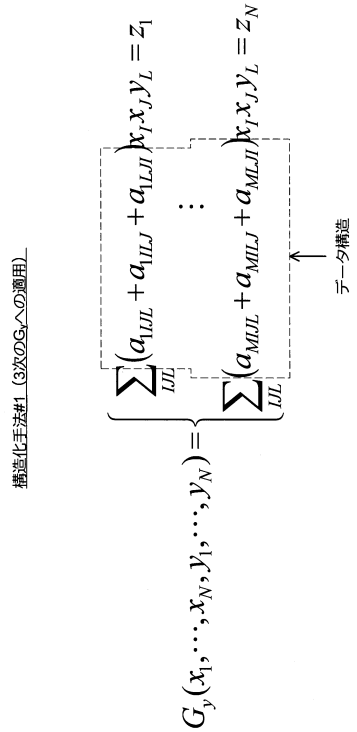
【図２４】



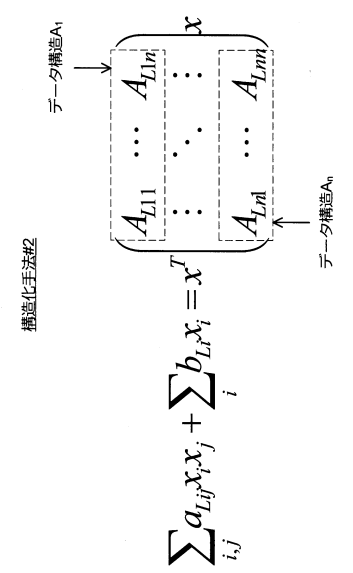
【図２５】



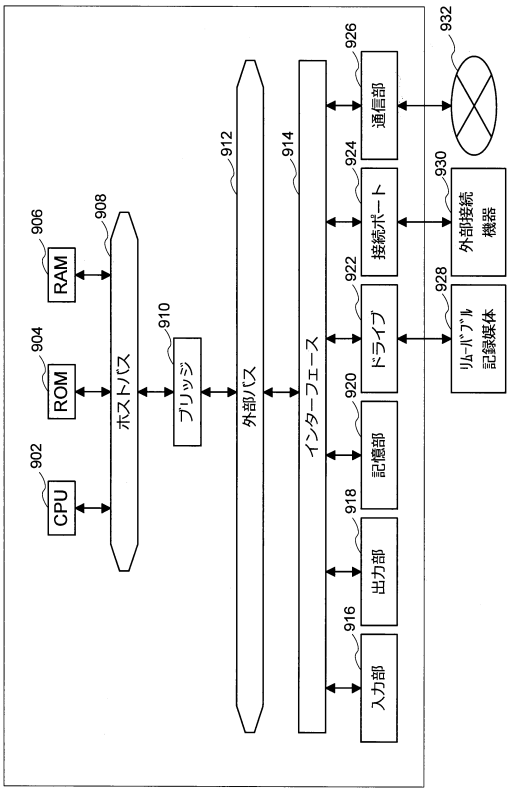
【図２６】



【図２７】



【図２８】


902
CPU
904
ROM
906
RAM
908
ホストバス
910
ブリッジ
912
外部バス
914
インターフェース
916
入力部
918
出力部
920
記憶部
922
ドライブ
924
接続ポート
926
通信部
928
リムーバブル記録媒体
930
外部接続機器
932

フロントページの続き

(72)発明者　樋渡　玄良
東京都港区港南１丁目７番１号　ソニー株式会社内

審査官　中里　裕正

(56)参考文献　特開２００４－２０１２９２（ＪＰ，Ａ）
特開２００４－１５８９５１（ＪＰ，Ａ）
Sakumoto, K., Shirai, T. and Hiwatari, H., Public-Key Identification Schemes Based on Multivariate Quadratic Polynomials, Lecture Notes in Computer Science, ２０１１年　８月１８日, Vol.6841, p.706-723
Hankerson, D., Menezes, A. and Vanstone, S., Guide to Elliptic Curve Cryptography, Springer-Verlag New York, Inc., ２００４年, p.47-61, 213-215
Berbain, C., Billet, O. and Gilbert, H., Efficient Implementations of Multivariate Quadratic Systems, Lecture Notes in Computer Science, ２００７年, Vol.4356, p.174-187
Chen, A. I.-T., et al., SSE Implementation of Multivariate PKCs on Modern x86 CPUs, Lecture Notes in Computer Science, ２００９年, Vol.5747, p.33-48

(58)調査した分野(Int.Cl., ＤＢ名)
Ｈ０４Ｌ　　９／３２
Ｇ０９Ｃ　　１／００
ＪＳＴＰｌｕｓ／ＪＭＥＤＰｌｕｓ／ＪＳＴ７５８０（ＪＤｒｅａｍＩＩＩ）
ＩＥＥＥ　Ｘｐｌｏｒｅ