(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第5272417号
(P5272417)
(45)発行日　平成25年8月28日(2013.8.28)　(24)登録日　平成25年5月24日(2013.5.24)

(51)Int.Cl.　F I
G09C 1/00 (2006.01)　G09C 1/00 610A

請求項の数 20　(全 56 頁)

(21)出願番号　特願2008-10548(P2008-10548)
(22)出願日　平成20年1月21日(2008.1.21)
(65)公開番号　特開2009-175167(P2009-175167A)
(43)公開日　平成21年8月6日(2009.8.6)
審査請求日　平成22年12月27日(2010.12.27)

(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　100093241
弁理士　宮田　正昭
(74)代理人　100101801
弁理士　山田　英治
(74)代理人　100086531
弁理士　澤田　俊夫
(74)代理人　100095496
弁理士　佐々木　榮二
(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

最終頁に続く

(54)【発明の名称】データ変換装置、およびデータ変換方法、並びにコンピュータ・プログラム

(57)【特許請求の範囲】
【請求項１】
データ変換装置であり、
ラウンド演算を繰り返すデータ変換処理を行なうデータ変換部を有し、
前記データ変換部は、
前記ラウンド演算において、
同一サイズのデータブロックを配列した長方形配列データを２分割した２つの分割データを生成し、
生成した２つの分割データに対して複数の演算を、順次、実行して各演算処理結果である更新分割データを逐次生成し、最終更新分割データをラウンド演算結果とする構成であり、
前記複数の演算は、
（ａ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした線形変換処理、
（ｂ）前記分割データと、更新分割データとの排他的論理和演算、または、更新分割データ間の排他的論理和演算、
（ｃ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象としたシフト処理、
（ｄ）前記分割データと、更新分割データとのスワップ処理、または、更新分割データ間のスワップ処理、

上記（ａ）～（ｄ）の各処理を含む演算であるデータ変換装置。
【請求項２】
前記同一サイズのデータブロックは１バイト単位のデータブロックであり、前記データ変換部は、前記ラウンド演算において、１バイト単位のデータブロックを配列した長方形配列データを２分割した分割データに対する処理を行う構成である請求項１に記載のデータ変換装置。
【請求項３】
前記データ変換部は、
前記ラウンド演算において、さらに、
（ｅ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした非線形変換処理、
（ｆ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした鍵適用演算処理を実行する構成である請求項１に記載のデータ変換装置。
【請求項４】
前記鍵適用演算は、前記分割データ、または更新分割データ中のいずれかの分割データの構成データと暗号鍵データとの排他的論理和演算である請求項３に記載のデータ変換装置。
【請求項５】
前記データ変換部は、
前記排他的論理和演算の結果を前記分割データ、または更新分割データの新たな更新データとして設定する請求項１に記載のデータ変換装置。
【請求項６】
前記データ変換部は、
前記シフト処理の実行に際して、
ｍ行２ｎ列の長方形配列データ中、シフト処理対象となるｍ行ｎ列の分割データ、または更新分割データが、ｍ≦ｎである場合、シフト前に同じ列のデータブロックがシフト処理後に異なる列になるようにシフトし、ｍ＞ｎの場合には、シフト前に同じ列のデータブロックがシフト処理後の任意の列に（ｍ／ｎ）－１個以上、（ｍ／ｎ）＋１個以下の範囲内で含まれるようにシフト処理を実行する請求項１に記載のデータ変換装置。
【請求項７】
前記データ変換部は、
前記シフト処理を２分割した分割データ、または更新分割データの双方に対して実行する請求項１に記載のデータ変換装置。
【請求項８】
前記データ変換部は、
前記ラウンド演算において、
前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行して分割データＡの更新を行い、さらに更新された分割データＡに対する線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行する請求項１に記載のデータ変換装置。
【請求項９】
前記データ変換部は、
前記ラウンド演算において、
前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行し、さらに線形変換処理を実行して分割データＡの更新を行い、さらに、更新された分割データＡと、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行する請求項１に記載のデータ変換装置。
【請求項１０】

前記データ変換部は、
前記ラウンド演算において、
前記分割データの一方の分割データＡに対して非線形変換処理と、線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対するシフト処理と鍵データとの排他的論理和処理を実行する請求項１に記載のデータ変換装置。
【請求項１１】
前記データ変換部は、
前記ラウンド演算において、
前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理と線形変換処理を実行し、さらに、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＡの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行する請求項１に記載のデータ変換装置。
【請求項１２】
前記データ変換部は、
前記ラウンド演算における線形変換処理において、複数の異なる行列をラウンド単位で選択適用する構成である請求項１に記載のデータ変換装置。
【請求項１３】
前記データ変換部は、
複数の異なる行列の選択適用としてＤＳＭ（Ｄｉｆｆｕｓｉｏｎ　Ｓｗｉｔｃｈｉｎｇ　Ｍｅｃｈａｎｉｓｍ）を利用した処理を行う構成である請求項１２に記載のデータ変換装置。
【請求項１４】
データ変換装置において実行するデータ変換方法であり、
データ変換部が、ラウンド演算を繰り返してデータ変換を行なうデータ変換ステップを有し、
前記データ変換ステップは、
前記ラウンド演算において、
同一サイズのデータブロックを配列した長方形配列データを２分割した２つの分割データを生成し、
生成した２つの分割データに対して複数の演算を、順次、実行して各演算処理結果である更新分割データを逐次生成し、最終更新分割データをラウンド演算結果とする処理を実行し、
前記複数の演算は、
（ａ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした線形変換処理、
（ｂ）前記分割データと、更新分割データとの排他的論理和演算、または、更新分割データ間の排他的論理和演算、
（ｃ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象としたシフト処理、
（ｄ）前記分割データと、更新分割データとのスワップ処理、または、更新分割データ間のスワップ処理、
上記（ａ）～（ｄ）の各処理を含む演算であるデータ変換方法。
【請求項１５】
前記同一サイズのデータブロックは１バイト単位のデータブロックであり、前記データ変換部は、前記ラウンド演算において、１バイト単位のデータブロックを配列した長方形配列データを２分割した分割データに対する処理を行う構成である請求項１４に記載のデータ変換方法。
【請求項１６】

前記データ変換ステップは、
前記ラウンド演算において、さらに、
（ｅ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした非線形変換処理、
（ｆ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした鍵適用演算処理を実行する構成である請求項１４に記載のデータ変換方法。
【請求項１７】
前記鍵適用演算は、前記分割データ、または更新分割データ中のいずれかの分割データの構成データと暗号鍵データとの排他的論理和演算である請求項１６に記載のデータ変換方法。
【請求項１８】
前記データ変換ステップは、
前記排他的論理和演算の結果を前記分割データ、または更新分割データの新たな更新データとして設定する請求項１４に記載のデータ変換方法。
【請求項１９】
データ変換装置においてデータ変換処理を実行させるコンピュータ・プログラムであり、
データ変換部に、ラウンド演算を繰り返してデータ変換を行なわせるデータ変換ステップを有し、
前記データ変換ステップは、
前記ラウンド演算において、
同一サイズのデータブロックを配列した長方形配列データを２分割した２つの分割データを生成し、
生成した２つの分割データに対して複数の演算を、順次、実行して各演算処理結果である更新分割データを逐次生成し、最終更新分割データをラウンド演算結果とする処理を実行し、
前記複数の演算は、
（ａ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした線形変換処理、
（ｂ）前記分割データと、更新分割データとの排他的論理和演算、または、更新分割データ間の排他的論理和演算、
（ｃ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象としたシフト処理、
（ｄ）前記分割データと、更新分割データとのスワップ処理、または、更新分割データ間のスワップ処理、
上記（ａ）～（ｄ）の各処理を含む演算であるコンピュータ・プログラム。
【請求項２０】
プログラムを実行するプロセッサと、
前記プログラムを保持するメモリと、
ラウンド演算を繰り返すデータ変換処理を行なうデータ変換部を有し、
前記データ変換部は、
前記ラウンド演算において、
同一サイズのデータブロックを配列した長方形配列データを２分割した２つの分割データを生成し、
生成した２つの分割データに対して複数の演算を、順次、実行して各演算処理結果である更新分割データを逐次生成し、最終更新分割データをラウンド演算結果とする処理を実行し、
前記複数の演算は、
（ａ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象とした線形変換処理、

（ｂ）前記分割データと、更新分割データとの排他的論理和演算、または、更新分割データ間の排他的論理和演算、
（ｃ）前記分割データ、または更新分割データ中のいずれかの分割データを処理対象としたシフト処理、
（ｄ）前記分割データと、更新分割データとのスワップ処理、または、更新分割データ間のスワップ処理、
上記（ａ）～（ｄ）の各処理を含む演算である情報処理装置。

【発明の詳細な説明】
【技術分野】
【０００１】
本発明は、データ変換装置、およびデータ変換方法、並びにコンピュータ・プログラムに関する。さらに詳細には、効率的なデータ拡散や暗号処理を実現するデータ変換装置、およびデータ変換方法、並びにコンピュータ・プログラムに関する。
【背景技術】
【０００２】
入力データに対してブロック単位のデータ変換処理を実行して暗号化を行なうブロック暗号や、ハッシュ関数などにおいては、入力データに対する高いデータ攪拌性能が求められる。例えば、入力データをバイト単位などの固定サイズに分割して線形変換や非線形変換などの様々な演算を繰り返し実行して、バイト単位データを互いに影響させながらデータを攪拌させるといった処理を行なう。
【０００３】
例えば米国標準暗号として知られているＡＥＳ（Advanced Encryption Standard）アルゴリズムは、入力データをバイト単位に分割し、バイト単位データを正方形や長方形の配列として配置し、行単位の処理や列単位の処理、具体的には非線形変換処理や線形変換処理などの様々な処理を繰り返すことで、データの攪拌を行なうアルゴリズムである。
【０００４】
具体例について図１を参照して説明する。変換処理の対象とするデータが８×１６＝１２８ビットデータである場合、図１（ａ）に示すように、a_1，a_2，a_3・・・a_{16}の各々を８ビットの１バイトデータとしたバイト単位データからなる正方形配列を設定して、
行単位の演算、例えば（a_1，a_2，a_3，a_4）など各行に対する演算処理、あるいは、
列単位の演算、例えば（a_1，a_5，a_9，a_{13}）など各列に対する演算、
このような様々なデータ単位に対する演算処理、具体的には、非線形変換処理や線形変換処理、シフト処理、鍵を適用した排他的論理和などの様々な処理を繰り返すことでデータ変換を行なう。
【０００５】
図１（ａ）に示すように、バイト単位データを正方形に配置して行単位あるいは列単位の処理を実行すると効率的な攪拌を実現することができることが知られている。しかし、１バイト単位データの正方形配列ができるのは、変換処理対象となる入力データが、図１（ａ）に示す８×１６＝１２８ビットデータのように特定ビット数のデータである場合に限られる。具体的には、
ビット数＝８×（ｎ）^{2}ビットである場合、（ｎは自然数）
バイト数として表現すると、
バイト数＝（ｎ）^{2}バイトである場合、（ｎは自然数）
このような場合に限られてしまう。
１２８ビットは、ビット数＝８×（４）^{2}ビットであり、図１（ａ）に示すように４×４の１６個のバイト単位データの正方形配列として設定できる。
【０００６】
しかし、変換対象となるデータが例えば２５６ビットである場合、

２５６＝８×３２
であり、
２５６＝８×（ｎ）^{2}ビット
として表現できないため、バイト単位の正方形配列は不可能となる。
このような場合、３２個の８ビットのバイト単位データａ$_{1}$，ａ$_{2}$，ａ$_{3}$・・・ａ$_{32}$の各々を図１（ｂ）に示すように、縦横比が１：２のような長方形に配置し、この長方形配列に対して、行単位の処理や列単位の処理を繰り返すことで攪拌を実行することになる。しかし、この図１（ｂ）に示すような長方形配列に対しては、正方形の場合と同様な手順で攪拌を行っても演算の手間が増える割に攪拌性能が上がらないという問題がある。

【０００７】
図２以下を参照して、バイト単位の単位データを正方形配列とした場合（正方ステート）と、長方形配列とした場合（長方形ステート）の攪拌処理例について説明する。

【０００８】
（Ａ）正方形配列（正方ステート）に対する処理例
図２以下を参照して１２８ビットのデータに対するデータ変換処理における攪拌処理について説明する。１２８ビットのデータはバイト（８ビット）単位のデータに分割される。ここでは分割された１バイト単位の１６個のデータをａ$_{1}$からａ$_{16}$で表す。

【０００９】
図２に示す正方形配列データ（正方ステート）１１にあるように１バイト単位の１６個のデータ［ａ$_{1}$～ａ$_{16}$］を４×４の配列上に格納する。この正方形の配列に格納された状態のデータを以下、正方ステートと呼ぶ。

【００１０】
ＡＥＳブロック暗号アルゴリズムでは、正方ステートに対する演算が複数定義されており、定義された演算を繰り返し適用することで暗号化を実現している。ＡＥＳで定義されている演算は、図２に示す以下の４種類である。
（１）非線形変換処理（ＳＵＢ）、
各バイトデータにバイト単位の非線形変換Ｓ（ｘ）を施して値を更新する演算、
図２（１）に示すように、変換処理後のバイト単位の出力ｂ$_{i}$と入力ａ$_{i}$の関係は、
ｂ$_{i}$＝Ｓ（ａ$_{i}$）
ｉ＝１，２，・・・，１６、
である。例えばＡＥＳ暗号においてはＳ－ｂｏｘを利用した非線形変換に対応する。

【００１１】
（２）シフト処理（ＳＨＩＦＴ）、
行ごとにローテーションシフト演算を施す処理。シフト量は行ごとに異なり、ＡＥＳの場合は、図２（２）に示すように、一行目ローテーションシフトなし、二行目は１バイト分右方向へローテーションシフト、三行目は２バイト分、四行目は３バイト分のローテーションシフトを行う。

【００１２】
（３）線形変換処理（ＭＡＴ）、
列ごとの４つのデータをベクトルとみなし４×４の行列［Ｍ］による演算を施して値を更新する演算である。
変換処理後のバイト単位の出力ｂ$_{i}$と入力ａ$_{i}$の関係は、
^{t}（ｂ$_{i}$，ｂ$_{i+4}$，ｂ$_{i+8}$，ｂ$_{i+12}$）＝Ｍ^{t}（ａ$_{i}$，ａ$_{i+4}$，ａ$_{i+8}$，ａ$_{i+12}$）
ｉ＝１，２，３，４、
である。なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。すなわち、上記式は、以下を意味する。

【００１３】

【数１】

$$\begin{pmatrix} b_i \\ b_{i+4} \\ b_{i+8} \\ b_{i+12} \end{pmatrix} = M \begin{pmatrix} a_i \\ a_{i+4} \\ a_{i+8} \\ a_{i+12} \end{pmatrix}$$

【００１４】

（４）鍵適用演算処理（ＫＡＤＤ）、

鍵スケジュール部から出力されたラウンド鍵［ｋ$_i$］を各バイトデータに排他的論理和を行う演算である。

変換処理後のバイト単位の出力ｂ$_i$と入力ａ$_i$の関係は、

ｂ$_i$＝ａ$_i$（ＸＯＲ）ｋ$_i$

ｉ＝１，２，・・・，１６、

である。なお、上記式において（ＸＯＲ）は排他的論理和演算を示している。

【００１５】

上記の各演算（１）～（４）を所定シーケンスで順次実行する演算の組み合わせによって、１つのラウンド演算が設定される。入力データに対して、ラウンド演算を繰り返し実行し出力データ、例えば暗号化データを生成して出力する。ラウンド演算は、図３に示すように、例えば、（１）非線形変換処理（ＳＵＢ）→（２）シフト処理（ＳＨＩＦＴ）→（３）線形変換処理（ＭＡＴ）→（４）鍵適用演算処理（ＫＡＤＤ）の順で実行する一連のデータ変換処理によって構成され、このラウンド演算を複数回繰り返して実行し、入力データを変換して出力データ、すなわち暗号化データを生成する。

【００１６】

図４は、正方ステートに対して、（１）非線形変換処理（ＳＵＢ）→（２）シフト処理（ＳＨＩＦＴ）→（３）線形変換処理（ＭＡＴ）→（４）鍵適用演算処理（ＫＡＤＤ）の順で実行するラウンド演算を第１～第３ラウンド（Ｒ１～Ｒ３）実行した場合のデータ攪拌例について説明する図である。

【００１７】

図４では、初期状態の正方ステート２１の左上端の１バイトのバイト単位データ３１が、各ラウンド演算の（１）非線形変換処理（ＳＵＢ）、（２）シフト処理（ＳＨＩＦＴ）、（３）線形変換処理（ＭＡＴ）、（４）鍵適用演算処理（ＫＡＤＤ）によって、正方ステートに含まれるどのバイト単位データに影響を与えているかを示している。すなわち各演算の実行により正方ステート内の１バイトのデータ３１の構成ビットの影響が他のバイト単位データに拡散していく様子を示している。

【００１８】

入力データの初期状態の正方ステート２１中の左上のバイト単位データ３１に着目する（黒でマーク）。第１ラウンド（Ｒ１）目の非線形変換処理（ＳＵＢ）、シフト処理（ＳＨＩＦＴ）までは、バイト単位データ３１は、正方ステート中の他のバイト単位データの演算結果に影響を与えることはない。

【００１９】

しかし第１ラウンド目の線形変換処理（ＭＡＴ）を通過すると正方ステート中の左端の列に含まれる４つのバイト単位データに影響する。この状態を左上のバイト単位データ３１の構成ビットの影響が、正方ステート中の左端の列に含まれる４つのバイト単位データに拡散したと呼ぶ。

【００２０】

その後、鍵適用演算（ＫＡＤＤ）、第２ラウンドの非線形変換処理（ＳＵＢ）まではそれ以上の影響が広がらないが、次のシフト処理（ＳＨＩＦＴ）により縦に並んでいた４つのバイトが横方向に拡散され、各列に、バイト単位データ３１の影響を受けたバイト単位データがひとつずつ存在する状態となる。

【００２１】

さらにその直後の線形変換処理（ＭＡＴ）により正方ステートを構成する１６バイトすべてに影響が伝わることになる。

【００２２】

このケースではラウンド演算の２ラウンド分の処理によって、一つのバイト単位データが、正方ステートを構成するすべてのバイト単位データに影響を与える。なお図４では、一例として左上のバイト単位データ３１の影響について説明したが、正方ステートの任意の位置のバイトデータも、同様の影響を他のバイト単位データに与え、２ラウンドで、各バイト単位データの影響が全ての他のバイト単位データに影響を与える、すなわち拡散する。高速で広範囲な拡散処理は、データ攪拌性能の高さを証明するものであり、暗号化データの秘匿性や効率評価の要素として利用される。

【００２３】

図４に示す例では、一つのバイト単位データが、正方ステートを構成するすべてのバイト単位データに影響を与えるまでに２ラウンドを要している。すべてのバイトに影響を与えるまでにかかった演算コストを見積もる。全体に影響を与えるためには２ラウンドかかっているため２回ずつの非線形変換処理（ＳＵＢ）、シフト処理（ＳＨＩＦＴ）、線形変換処理（ＭＡＴ）、鍵適用演算処理（ＫＡＤＤ）が必要である。

【００２４】

一つの指標として演算に必要なハードウェアゲート規模がその演算の本質的な複雑度を表していると捉えるものとする。この場合シフト処理（ＳＨＩＦＴ）演算は回路の接続のみで実現することができるのでゲートを通過する必要がなく演算コスト＝０としてよい。

【００２５】

従って、図４に示す正方ステートにおいて、１つのバイト単位データが、正方ステートを構成するすべてのバイト単位データに影響を与えるまでの２ラウンド演算に必要とする演算コストは、

２ＳＵＢ＋２ＭＡＴ＋２ＫＡＤＤ

と見積もることができる。

なお、これらの演算処理を実行するためには、論理回路や演算処理プログラムなどが利用され、その構成によって必要とする演算回路や処理速度も異なることになる。従って絶対的な効率の評価は難しいが、上記の演算に必要となる論理回路におけるゲート数を１つの評価指標とすることが可能である。

ある論理回路実装例として、各演算に必要とするゲート数は、

ＳＵＢ演算＝３，２００～４，８００ゲート程度、

ＭＡＴ演算＝８００～１，２００ゲート程度、

ＫＡＤＤ演算＝３２０ゲート程度、

これらのゲート数に相当する。

【００２６】

従って、図４に示す例において、正方ステートを構成するすべてのバイト単位データに影響を与えるまでの２ラウンド演算に必要とする演算コストは、

２ＳＵＢ＋２ＭＡＴ＋２ＫＡＤＤ＝９，０００～１３，０００ゲート＝９Ｋゲート～１３Ｋゲート

の計算コストであると算出できる。

この計算コストが小さいほど、暗号処理やハッシュ処理などを実行する装置に必要とする回路規模の小型化が可能であり、また高速処理も可能となる。

【００２７】

（Ｂ）長方形配列（長方形ステート）に対する処理例

次に、図５以下を参照して２５６ビットのデータに対するデータ変換処理における攪拌処理について説明する。以下、ＡＥＳと同様の設計方針を持つアルゴリズム［Ｒｉｊｎｄａｅｌ］における変換処理における攪拌例について説明する。

【００２８】

２５６ビットのデータはバイト（８ビット）単位のデータに分割される。ここでは分割された１バイト単位の３２個のデータをa_1からa_{32}で表す。図５に示す長方形配列データ（長方形ステート）５１にあるように１バイト単位の３２個のデータ［a_1～a_{32}］を４×８の配列上に格納する。この長方形の配列に格納された状態のデータを以下、長方形ステートと呼ぶ。

【００２９】

［Ｒｉｊｎｄａｅｌ］アルゴリズムでは、先に図２～図４を参照して説明した正方ステートで利用された非線形変換処理（ＳＵＢ）、シフト処理（ＳＨＩＦＴ）、線形変換処理（ＭＡＴ）、鍵適用演算処理（ＫＡＤＤ）を、長方形ステートに適用させるために拡張された演算を以下のように定義する。

【００３０】

［Ｒｉｊｎｄａｅｌ］アルゴリズムで、定義されている演算は、図５に示す以下の４種類である。

（１）非線形変換処理（Ｗ－ＳＵＢ）、

各バイトデータにバイト単位の非線形変換Ｓ（ｘ）を施して値を更新する演算、

図５（１）に示すように、変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$b_i = S(a_i)$

$i = 1, 2, \cdots, 32$、

である。

【００３１】

（２）シフト処理（Ｗ－ＳＨＩＦＴ）、

行ごとにローテーションシフト演算を施す処理。シフト量は行ごとに異なり、Ｒｉｊｎｄａｅｌの場合は、図５（２）に示すように、一行目ローテーションシフトなし、二行目は１バイト分右方向へローテーションシフト、三行目は３バイト分、四行目は４バイト分のローテーションシフトを行う。

【００３２】

（３）線形変換処理（Ｗ－ＭＡＴ）、

列ごとの４つのデータをベクトルとみなし４×４の行列［Ｍ］による演算を施して値を更新する演算である。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = M\,^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$

$i = 1, 2, 3, \cdots 8$、

である。なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【００３３】

（４）鍵適用演算処理（Ｗ－ＫＡＤＤ）、

鍵スケジュール部から出力されたラウンド鍵［k_i］を各バイトデータに排他的論理和を行う演算である。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$b_i = a_i$（ＸＯＲ）k_i

$i = 1, 2, \cdots, 32$、

である。なお、上記式において（ＸＯＲ）は排他的論理和演算を示している。

【００３４】

上記の各演算（１）～（４）を所定シーケンスで順次実行する演算の組み合わせによって、１つのラウンド演算が設定される。入力データに対して、ラウンド演算を繰り返し実行し出力データ、例えば暗号化データを生成して出力する。ラウンド演算は、図６に示す

ように、例えば、（１）非線形変換処理（Ｗ－ＳＵＢ）→（２）シフト処理（Ｗ－ＳＨＩＦＴ）→（３）線形変換処理（Ｗ－ＭＡＴ）→（４）鍵適用演算処理（Ｗ－ＫＡＤＤ）の順で実行する一連のデータ変換処理によって構成され、このラウンド演算を複数回繰り返して実行し、入力データを変換して出力データ、すなわち暗号化データを生成する。

【００３５】

図７は、長方形ステートに対して、（１）非線形変換処理（Ｗ－ＳＵＢ）→（２）シフト処理（Ｗ－ＳＨＩＦＴ）→（３）線形変換処理（Ｗ－ＭＡＴ）→（４）鍵適用演算処理（Ｗ－ＫＡＤＤ）の順で実行するラウンド演算を第１～第３ラウンド（Ｒ１～Ｒ３）実行した場合のデータ攪拌例について説明する図である。

【００３６】

入力データの初期状態の長方形ステート６１中の左上のバイト単位データ７１に着目する（黒でマーク）。前述の正方ステートのケースと同様に、２ラウンド後には１６バイトのデータに影響をおよぼしていることがわかる。さらに３ラウンド目のシフト処理（Ｗ－ＳＨＩＦＴ）により影響範囲が拡散され、直後の、３ラウンド目の線形変換処理（Ｗ－ＭＡＴ）により３２バイトすべてに影響を及ぼしていることがわかる。

【００３７】

このケースではラウンド演算の３ラウンド分の処理によって、一つのバイト単位データが、長方形ステートを構成するすべてのバイト単位データに影響を与える。なお図７では、一例として左上のバイト単位データ７１の影響について説明したが、この長方形ステートの任意の位置のバイトデータも、同様の影響を他のバイト単位データに与え、３ラウンドで、各バイト単位データの影響が全ての他のバイト単位データに影響を与える、すなわち拡散する。

【００３８】

次に、先の正方ステート（図４）の例と同様、ゲート数による演算コストを算出する。図７に示す例において、長方形ステートを構成するすべてのバイト単位データに影響を与えるまでの３ラウンド演算に必要とする演算コストは、３回ずつのＷ－ＳＵＢ，Ｗ－ＳＨＩＦＴ，Ｗ－ＭＡＴ，Ｗ－ＫＡＤＤ演算が必要である。なお前述したようにシフト処理（Ｗ－ＳＨＩＦＴ）は演算コスト＝０とすることが可能であり、従ってこの場合の演算コストは、

３（Ｗ－ＳＵＢ）＋３（Ｗ－ＭＡＴ）＋３（Ｗ－ＫＡＤＤ）と見積もることができる。

Ｗ－ＳＵＢ，Ｗ－ＭＡＴ，Ｗ－ＫＡＤＤは、それぞれＳＵＢ，ＭＡＴ，ＫＡＤＤの２倍のコストに相当する。従って、この長方形ステートにおいて、長方形ステートを構成するすべてのバイト単位データに影響を与えるまでの３ラウンド演算に必要とする演算コストは、先の［００２５］において説明したゲート数に基づいて算出すると、２６Ｋゲート～３８Ｋゲートとなる。

【００３９】

上述したように、図２～図４を参照して説明したように、データをバイト単位データとして正方形に配置してラウンド演算を行なう場合には、比較的低い計算コストでの攪拌が達成できるが、図５～図７を参照して説明したような、正方形配列ができない２５６ビット等の入出力に対応するために設定される長方形配列を利用した処理では、演算コストが増大してしまうという問題がある。

【発明の開示】

【発明が解決しようとする課題】

【００４０】

本発明は、上記問題点に鑑みてなされたものであり、長方形配列によるデータ攪拌を実行する暗号処理やハッシュ処理、データ拡散処理などにおいて、演算コストを低下させ効率的な拡散を実現するデータ変換装置、およびデータ変換方法、並びにコンピュータ・プログラムを提供することを目的とする。

【課題を解決するための手段】

【００４１】

本発明の第１の側面は、
データ変換装置であり、
ラウンド演算を繰り返すデータ変換処理を行なうデータ変換部を有し、
前記データ変換部は、
前記ラウンド演算において、
同一サイズのデータブロックを配列した長方形配列データを２分割した分割データの一方に対する線形変換処理と、
２つの分割データ相互の排他的論理和演算処理と、
分割データの一方のデータに対するシフト処理と、
２つの分割データのスワップ処理を実行する構成であることを特徴とするデータ変換装置にある。

【００４２】
さらに、本発明のデータ変換装置の一実施態様において、前記同一サイズのデータブロックは１バイト単位のデータブロックであり、前記データ変換部は、前記ラウンド演算において、１バイト単位のデータブロックを配列した長方形配列データを２分割した分割データに対する処理を行う構成であることを特徴とする。

【００４３】
さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記ラウンド演算において、さらに、前記分割データの一方に対する非線形変換処理と、前記分割データの一方に対する鍵適用演算処理を実行する構成であることを特徴とする。

【００４４】
さらに、本発明のデータ変換装置の一実施態様において、前記鍵適用演算は、前記分割データの一方の構成データと暗号鍵データとの排他的論理和演算であることを特徴とする。

【００４５】
さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記排他的論理和演算の結果を前記分割データの一方の更新データとして設定することを特徴とする。

【００４６】
さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記シフト処理の実行に際して、ｍ行２ｎ列の長方形配列データ中、シフト処理対象となるｍ行ｎ列の分割データが、ｍ≦ｎである場合、シフト前に同じ列のデータブロックがシフト処理後に異なる列になるようにシフトし、ｍ＞ｎの場合には、シフト前に同じ列のデータブロックがシフト処理後の任意の列に（ｍ／ｎ）－１個以上、（ｍ／ｎ）＋１個以下の範囲内で含まれるようにシフト処理を実行することを特徴とする。

【００４７】
さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記シフト処理を２分割した分割データの双方に対して実行することを特徴とする。

【００４８】
さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行して分割データＡの更新を行い、さらに更新された分割データＡに対する線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行することを特徴とする。

【００４９】
さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行し、さらに線形変換処理を実行して分割データＡの更新を行い、さらに、更新された分割データＡと、他方の分割データＢとの排他的論理和を実行して、その結

果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行することを特徴とする。

【００５０】

さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対するシフト処理と鍵データとの排他的論理和処理を実行することを特徴とする。

【００５１】

さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理と線形変換処理を実行し、さらに、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＡの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行することを特徴とする。

【００５２】

さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、前記ラウンド演算における線形変換処理において、複数の異なる行列をラウンド単位で選択適用する構成であることを特徴とする。

【００５３】

さらに、本発明のデータ変換装置の一実施態様において、前記データ変換部は、複数の異なる行列の選択適用としてＤＳＭ（Ｄｉｆｆｕｓｉｏｎ　Ｓｗｉｔｃｈｉｎｇ　Ｍｅｃｈａｎｉｓｍ）を利用した処理を行う構成であることを特徴とする。

【００５４】

さらに、本発明の第２の側面は、

データ変換装置において実行するデータ変換方法であり、

データ変換部が、ラウンド演算を繰り返してデータ変換を行なうデータ変換ステップを有し、

前記データ変換ステップは、

前記ラウンド演算において、

同一サイズのデータブロックを配列した長方形配列データを２分割した分割データの一方に対する線形変換処理と、

２つの分割データ相互の排他的論理和演算処理と、

分割データの一方のデータに対するシフト処理と、

２つの分割データのスワップ処理を実行することを特徴とするデータ変換方法にある。

【００５５】

さらに、本発明のデータ変換方法の一実施態様において、前記同一サイズのデータブロックは１バイト単位のデータブロックであり、前記データ変換部は、前記ラウンド演算において、１バイト単位のデータブロックを配列した長方形配列データを２分割した分割データに対する処理を行う構成であることを特徴とする。

【００５６】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記ラウンド演算において、さらに、前記分割データの一方に対する非線形変換処理と、前記分割データの一方に対する鍵適用演算処理を実行する構成であることを特徴とする。

【００５７】

さらに、本発明のデータ変換方法の一実施態様において、前記鍵適用演算は、前記分割データの一方の構成データと暗号鍵データとの排他的論理和演算であることを特徴とする。

【００５８】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記排他的論理和演算の結果を前記分割データの一方の更新データとして設定することを特徴とする。

【００５９】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記シフト処理の実行に際して、ｍ行２ｎ列の長方形配列データ中、シフト処理対象となるｍ行ｎ列の分割データが、ｍ≦ｎである場合、シフト前に同じ列のデータブロックがシフト処理後に異なる列になるようにシフトし、ｍ＞ｎの場合には、シフト前に同じ列のデータブロックがシフト処理後の任意の列に（ｍ／ｎ）－１個以上、（ｍ／ｎ）＋１個以下の範囲内で含まれるようにシフト処理を実行することを特徴とする。

【００６０】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記シフト処理を２分割した分割データの双方に対して実行することを特徴とする。

【００６１】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行して分割データＡの更新を行い、さらに更新された分割データＡに対する線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行することを特徴とする。

【００６２】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行し、さらに線形変換処理を実行して分割データＡの更新を行い、さらに、更新された分割データＡと、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行することを特徴とする。

【００６３】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対するシフト処理と鍵データとの排他的論理和処理を実行することを特徴とする。

【００６４】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記ラウンド演算において、前記分割データの一方の分割データＡに対して非線形変換処理と、シフト処理と線形変換処理を実行し、さらに、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＡの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行することを特徴とする。

【００６５】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、前記ラウンド演算における線形変換処理において、複数の異なる行列をラウンド単位で選択適用する構成であることを特徴とする。

【００６６】

さらに、本発明のデータ変換方法の一実施態様において、前記データ変換ステップは、複数の異なる行列の選択適用としてＤＳＭ（Ｄｉｆｆｕｓｉｏｎ　Ｓｗｉｔｃｈｉｎｇ　Ｍｅｃｈａｎｉｓｍ）を利用した処理を行う構成であることを特徴とする。

【００６７】
さらに、本発明の第３の側面は、
データ変換装置においてデータ変換処理を実行させるコンピュータ・プログラムであり、
データ変換部に、ラウンド演算を繰り返してデータ変換を行なわせるデータ変換ステップを有し、
前記データ変換ステップは、
前記ラウンド演算において、
同一サイズのデータブロックを配列した長方形配列データを２分割した分割データの一方に対する線形変換処理と、
２つの分割データ相互の排他的論理和演算処理と、
分割データの一方のデータに対するシフト処理と、
２つの分割データのスワップ処理を実行するステップであることを特徴とするコンピュータ・プログラムにある。
【００６８】
なお、本発明のプログラムは、例えば、様々なプログラム・コードを実行可能な汎用システムに対して、コンピュータ可読な形式で提供する記憶媒体、通信媒体によって提供可能なプログラムである。このようなプログラムをコンピュータ可読な形式で提供することにより、コンピュータ・システム上でプログラムに応じた処理が実現される。
【００６９】
本発明のさらに他の目的、特徴や利点は、後述する本発明の実施例や添付する図面に基づくより詳細な説明によって明らかになるであろう。なお、本明細書においてシステムとは、複数の装置の論理的集合構成であり、各構成の装置が同一筐体内にあるものには限らない。
【発明の効果】
【００７０】
本発明の一実施例によれば、例えば１バイト単位のデータブロックを配列した長方形配列データを２分割して設定した分割データに対する様々な処理を実行してデータ変換を行う構成において、分割データの一方に対する線形変換処理と、２つの分割データ相互の排他的論理和演算処理と、分割データの一方のデータに対するシフト処理と、２つの分割データのスワップ処理を実行することで、演算コストを低減した効率的なデータ攪拌を実現することができる。さらに、分割データに対する非線形変換や鍵適用演算を含めることでセキュリティレベルの高い暗号処理が実現される。
【発明を実施するための最良の形態】
【００７１】
以下、図面を参照しながら本発明のデータ変換装置、およびデータ変換方法、並びにコンピュータ・プログラムの詳細について説明する。
【００７２】
本発明は、例えば入出力が２５６ビットなどのデータ、すなわち、８ビットのバイト単位データの配列を行なった場合、正方形の配列が出来ないビット数を持つデータ、すなわち、具体的には、
ビット数＝８×（ｎ）^{2}ビットである場合、（ｎは自然数）
バイト数として表現すると、
バイト数＝（ｎ）^{2}バイトである場合、（ｎは自然数）
このようなビット数（またはバイト数）とならないデータに対するデータ攪拌処理を効率的に実行する構成を提案するものである。
【００７３】
先に図１～図７を参照して説明したように、正方形の配列が出来ないビット数を持つデータについては図１（ｂ）に示すような長方形配列を行なって処理を行なうことになる。本発明では、このような長方形ステートを適用した処理において、演算コストを低減させ

、かつ十分な攪拌性能を実現させる手法を提案する。

【００７４】

例えば１つの具体例は、同一サイズのデータブロック、例えばバイト単位データを配列した長方形配列データを２分割した分割データの一方の分割データのみに非線形変換、線形変換、鍵の加算などの演算を行い、さらに、演算コストの低い排他的論理和（ＸＯＲ）演算や、スワップ（ＳＷＡＰ）演算などを行なう設定とする。この様な構成とすることで少ない処理量で十分な攪拌性能を実現する。本発明において提案する構成を適用することで、演算負荷を低減させて十分な拡散性能が得られ、暗号処理、ハッシュ処理、攪拌処理などを小さな回路規模で高速に行なうことが可能となる。

【００７５】

先に説明したように、ＡＥＳや、Ｒｉｊｎｄａｅｌなどのアルゴリズムでは、複数の演算の組み合わせからなるラウンド演算を繰り返し実行する。ラウンド演算には、非線形変換処理（ＳＵＢ，Ｗ－ＳＵＢ）、シフト処理（ＳＨＩＦＴ，Ｗ－ＳＨＩＦＴ）、線形変換処理（ＭＡＴ，Ｗ－ＭＡＴ）、鍵適用演算処理（ＫＡＤＤ，Ｗ－ＫＡＤＤ）が含まれる。

【００７６】

これらの処理中、非線形変換処理（ＳＵＢ，Ｗ－ＳＵＢ）と、線形変換処理（ＭＡＴ，Ｗ－ＭＡＴ）は、比較的重い処理、すなわち演算コストの高い処理である。従って、これらのコストの高い処理を減少させることができれば、全体の演算コストの低減が可能となると考えられる。

【００７７】

先に説明したＲｉｊｎｄａｅｌアルゴリズムは、長方形ステートのすべてのデータに対して、非線形変換処理（Ｗ－ＳＵＢ）や線形変換処理（Ｗ－ＭＡＴ）などを行なうアルゴリズムであるが、本発明の１つの実施例では、長方形ステートのすべてのデータに対して、このような設定とせず、長方形ステートを２分割した分割データの一方の分割データのみに、非線形変換処理や線形変換処理や鍵適用演算処理を行い、シフト処理やスワップ演算など、比較的処理コストの低い演算により、その影響を残り野データに拡散させる構成として十分な拡散性を達成する。

【００７８】

なお、本明細書では、バイト単位データによって構成される長方形配列データ（長方形ステート）のすべてのデータに対する処理と、長方形配列データ（長方形ステート）を２分割した分割データの一方の分割データに対する処理を以下のような表記によって区別して説明する。長方形ステートのすべてのデータに対する処理は、例えば、非線形変換処理（Ｗ－ＳＵＢ）、線形変換処理（Ｗ－ＭＡＴ）など、［Ｗ－］を設定した表記として示し、半分の分割データに対する処理は、非線形変換処理（Ｈ－ＳＵＢ）、線形変換処理（Ｈ－ＭＡＴ）など、［Ｈ－］を設定した表記として示す。

【００７９】

図８に、４×８＝３２のバイト単位データ（２５６ビット）によって構成される長方形ステートを２分割して設定した分割データに対する以下の処理、すなわち、

（１）非線形変換処理（Ｈ－ＳＵＢ）

（２）線形変換処理（Ｈ－ＭＡＴ）

（３）鍵適用演算（Ｈ－ＫＡＤＤ）

の処理例を示す。なお、図８では、長方形ステートの左半分の分割データに対してのみ上記各演算を適用した例である。なお、演算の適用対象は左半分または右半分、いずれでも同様の演算削減効果が発揮される。

【００８０】

図８に示す各演算の詳細は以下の通りである。

（１）非線形変換処理（Ｈ－ＳＵＢ）、

長方形ステートの左半分の分割データ各バイトデータにバイト単位の非線形変換Ｓ（ｘ）を施して値を更新する演算、

図８（１）に示すように、変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$b_i = S(a_i)$, if $i = 1 \sim 4, 9 \sim 12, 17 \sim 20, 25 \sim 28$,
$b_i = a_i$, else
上記のように右半分の分割データは変更しない。

【００８１】

（２）線形変換処理（Ｈ－ＭＡＴ）、
長方形ステートの左半分の分割データ各バイトデータについて、列ごとの４つのデータをベクトルとみなし４×４の行列［Ｍ］による演算を施して値を更新する演算である。
変換処理後のバイト単位の出力b_iと入力a_iの関係は、
$^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = M\,^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$
$i = 1, 2, 3, 4$、
$^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = {}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$
$i = 5, 6, 7, 8$、
上記のように、長方形ステートの左半分の各バイトデータについて行列［Ｍ］を適用した線形変換を実行し、右半分の分割データは変更しない。
なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【００８２】

（３）鍵適用演算処理（Ｈ－ＫＡＤＤ）、
長方形ステートの左半分の分割データ各バイトデータについて、鍵スケジュール部から出力されたラウンド鍵［k_i］を各バイトデータに排他的論理和を行う演算である。
変換処理後のバイト単位の出力b_iと入力a_iの関係は、
$b_i = a_i$（ＸＯＲ）k_i, if $i = 1 \sim 4, 9 \sim 12, 17 \sim 20, 25 \sim 28$,
$b_i = a_i$, else
上記のように右半分の分割データは変更しない。
である。なお、上記式において（ＸＯＲ）は排他的論理和演算を示している。

【００８３】

このように、長方形ステートの左半分の分割データ各バイトデータについて、非線形変換処理（Ｈ－ＳＵＢ）、線形変換処理（Ｈ－ＭＡＴ）、鍵適用演算（Ｈ－ＫＡＤＤ）を実行し、それ以外のデータに対してはシフト、スワップなどの比較的処理コストの低い演算を組み合わせて十分な拡散性を達成して性能の高い置換関数を実現する。なお、通常の暗号アルゴリズムでは、鍵適用演算処理（Ｈ－ＫＡＤＤ）は、非線形変換処理（Ｈ－ＳＵＢ）の処理サイズにあわせて定義されるのが一般的である。

【００８４】

目標としては、例えば、図８に示すような４×８＝３２のバイト単位データ（２５６ビット）に対してラウンド演算を繰り返し実行するデータ変換や暗号処理において、長方形ステートの半分のバイト単位データに対する演算である非線形変換処理（Ｈ－ＳＵＢ）や、線形変換処理（Ｈ－ＭＡＴ）を各ラウンドで１回ずつ実行する構成とした場合、５ラウンド分のラウンド演算ですべての３２バイトデータに影響を与える構成にできることが望ましい。

【００８５】

なぜなら、先に、図５～図７を参照して説明した長方形ステートの全体に対して、非線形変換処理（Ｗ－ＳＵＢ）や、線形変換処理（Ｗ－ＭＡＴ）を実行する構成においては、３ラウンドのラウンド演算で、１つのバイト単位データが、長方形ステートのすべてのバイト単位データに影響をさせることができるため、半分のコストとみなせる非線形変換処理（Ｈ－ＳＵＢ）や、線形変換処理（Ｈ－ＭＡＴ）を適用して２倍の６ラウンドで影響させることができても効率的な優位性がなくなるためである。従って５ラウンド以内で、１つのバイト単位データが、長方形ステートのすべてのバイト単位データに影響をさせることができる構成が求められる。

【００８６】

まず、上述した長方形ステートを２分割して設定した分割データに対する処理、すなわち、
（１）非線形変換処理（Ｈ－ＳＵＢ）
（２）線形変換処理（Ｈ－ＭＡＴ）
（３）鍵適用演算（Ｈ－ＫＡＤＤ）
これらの処理と、長方形ステートの全体データに対するシフト処理（Ｗ－ＳＨＩＦＴ）を組み合わせて、１ラウンドのラウンド演算を行なう設定とした場合のデータ拡散について考察する。

【００８７】

すなわち、図９に示すように、長方形ステート（２ｎ列×ｍ行）のｎｍ個のバイト単位データ（ただし２ｎ≠ｍ）からなる長方形ステートに対して、
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）
（２）長方形ステートの全体データに対するシフト処理（Ｗ－ＳＨＩＦＴ）
（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）
（４）長方形ステートの半分の分割データに対する鍵適用演算（Ｈ－ＫＡＤＤ）
これらの（１）～（４）の処理を１ラウンド分の演算として設定したラウンド演算を実行するものとする。

【００８８】

なお、シフト処理（Ｗ－ＳＨＩＦＴ）は、先に説明した［Ｒｉｊｎｄａｅｌ］アルゴリズムのシフト処理（Ｗ－ＳＨＩＦＴ）と同じ処理であり、行ごとにローテーションシフト演算を施す処理であり、図５（２）に示したように、一行目ローテーションシフトなし、二行目は１バイト分右方向へローテーションシフト、三行目は３バイト分、四行目は４バイト分のローテーションシフトを行う処理とする。

【００８９】

このようなラウンド演算を実行した場合の、データ拡散例について図１０を参照して説明する。入力データの初期状態の長方形ステート１００中の左上のバイト単位データ１０１に着目する（黒でマーク）。図から理解されるように、ラウンド演算の２ラウンド実行後には、左半分の分割データの３／４のバイト単位データと、右半分の分割データの１つのバイト単位データに長方形ステート１００中の左上のバイト単位データ１０１の構成ビットの影響が発生、すなわち拡散し、３ラウンド後には、左半分の分割データの全てのバイト単位データと、右半分の分割データの７つのバイト単位データに影響を与えている。

【００９０】

しかし、この３ラウンドでは、入力データの初期状態の長方形ステート１００中の左上のバイト単位データ１０１が長方形ステート１００の全てのバイト単位データに対して影響を与えるまでには至っていない。

【００９１】

次に、同様の処理における長方形ステート１００中の右半分の分割データの左上のバイト単位データ、すなわち、最上位の行の左から５番目のデータについての影響について図１１に示す。

【００９２】

入力データの初期状態の長方形ステート１００中の最上位の行の左から５番目のバイト単位データ１０２に着目する（黒でマーク）。図から理解されるように、ラウンド演算の３ラウンド実行後まで、全く他のバイト単位データ１０２は、他のデータに影響を与えない。右半分の分割データに対して実行される処理はシフト処理（Ｗ－ＳＨＩＦＴ）のみであるが、最上位の行にはシフトが施されないため、移動することがないためである。

【００９３】

かつ右半分の分割データにデータが位置する限り、非線形変換処理（Ｈ－ＳＵＢ）、線形変換処理（Ｈ－ＭＡＴ）、鍵適用演算（Ｈ－ＫＡＤＤ）、これらの演算対象として選択されないため影響範囲が広がることがない。従って、ラウンド処理を永遠に繰り返しても他のバイトに影響することがない。これは暗号処理に用いる関数としては望ましくない性

質である。

【００９４】

このように、図９に示すような、

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）

（２）長方形ステートの全体データに対するシフト処理（Ｗ－ＳＨＩＦＴ）

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データに対する鍵適用演算（Ｈ－ＫＡＤＤ）

これらの（１）～（４）の処理を１ラウンド分の演算として設定したラウンド演算を実行する構成では、十分な拡散が実行されず、暗号処理やハッシュ処理、データ拡散処理などに適用するには好ましくない。

【００９５】

以下、このような問題点に鑑み、少ないラウンド数で十分な拡散性能を持つ本発明に従ったアルゴリズムについて説明する。

【００９６】

［実施例１］

本発明の第１実施例におけるラウンド演算の構成を図１２に示す。実施例１のラウンド演算の処理シーケンスは、以下の通りである。

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理を行い、残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）

（４）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（５）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（５）の一連の処理をラウンド演算とする。

【００９７】

この実施例１のラウンド演算処理を要約すると、データ変換装置のデータ変換部は以下のラウンド演算を実行することになる。長方形ステートを２分割した分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行して分割データＡの更新を行い、さらに更新された分割データＡに対する線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行する。このようなラウンド処理を実行する。

【００９８】

なお、実施例においては長方形ステートの左半分の分割データを選択して非線形変換処理（Ｈ－ＳＵＢ）などの主要演算処理を実行する構成として説明するが、主要演算を実行する対象は右半分とする構成としてもよい。すなわち、以下で説明する処理において、左右のデータを入れ替えた処理を行なっても効果は同様である。

【００９９】

各処理の詳細について説明する。

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

この処理は、先に図８を参照して説明した処理と同様の処理であり、長方形ステートの左半分の各バイトデータにバイト単位の非線形変換Ｓ（ｘ）を施して値を更新する演算である。

図８（１）に示すように、変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$b_i = S(a_i)$，ｉｆ＝ｉ＝１～４，９～１２，１７～２０，２５～２８，

$b_i = a_i$，ｅｌｓｅ

上記のように右半分の分割データは変更しない。

【０１００】

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

このシフト処理（Ｈ－ＳＨＩＦＴ）について、図１３（Ａ）を参照して説明する。長方形ステートの左半分の分割データに対してのみ、各行について異なるシフト量（０～（ｎ－１））のシフトを実行する。例えば、先に図２を参照して説明した正方ステートに対するシフト処理と同様、一行目ローテーションシフトなし、二行目は１バイト分右方向へローテーションシフト、三行目は２バイト分、四行目は３バイト分のローテーションシフトを行う。

【０１０１】

（３）長方形ステートの半分の分割データに対する線形変換処理を行い、残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）

この線形変換および排他的論理和演算（ＭＡＴ－ＸＯＲ）について図１３（Ｂ）を参照して説明する。

【０１０２】

長方形ステートの左半分にある列ごとにデータをベクトルとみなして、予め設定した線形変換行列［Ｍ］を適用して４×４の行列演算を施して、その結果を右半分の対応する列のデータと排他的論理和（ＸＯＲ）を行い、その結果を右半分の分割データとして更新する。左半分の分割データは更新しない。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

（３－１）左半分の分割データ

$^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = {}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$

ｉ＝１，２，３，４、

（３－２）右半分の分割データ

$^t(b_{i+4}, b_{i+12}, b_{i+20}, b_{i+28}) = [M\,{}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})]\ (XOR)\ {}^t(a_{i+4}, a_{i+12}, a_{i+20}, a_{i+28})$

ｉ＝１，２，３，４、

である。なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【０１０３】

すなわち、左半分の分割データの左から第１列データと予め設定した線形変換行列［Ｍ］を適用して４×４の行列演算を施して、その結果と、右半分の分割データの左からの第１列データ（トータルでは左から５番目）との排他的論理和（ＸＯＲ）を実行した結果を、右半分の左からの第１列データ（トータルでは左から５番目）として更新する。以下、同様に、左半分の分割データの左から第２列データと予め設定した線形変換行列［Ｍ］を適用して４×４の行列演算を施して、その結果と、右半分の分割データの左からの第２列データ（トータルでは左から６番目）との排他的論理和（ＸＯＲ）を実行した結果を、右半分の左からの第２列データ（トータルでは左から６番目）の更新データとする。以下同様の処理を行なう。

【０１０４】

（４）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

このスワップ処理（ＳＷＡＰ）について図１３（Ｃ）を参照して説明する。この処理は、図に示すように、長方形ステートにおける左半分と右半分を入れ替える処理である。

【０１０５】

（５）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

この処理は、先に図８（３）を参照して説明した処理と同様の処理である。長方形ステートの左半分の各バイトデータについて、鍵スケジュール部から出力されたラウンド鍵［k_i］を各バイトデータに排他的論理和を行う。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$b_i = a_i$（ＸＯＲ）k_i，ｉｆ　ｉ＝１～４，９～１２，１７～２０，２５～２８，
$b_i = a_i$，ｅｌｓｅ
上記のように右半分の分割データは変更しない。
である。なお、上記式において（ＸＯＲ）は排他的論理和演算を示している。

【０１０６】
本発明の第１実施例におけるラウンド演算は、図１２に示すように、
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（３）長方形ステートの半分の分割データに対する線形変換処理を行い、残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）
（４）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（５）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの（１）～（５）の一連の処理によって構成される。すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行する。

【０１０７】
このラウンド演算を繰り返し実行することで、入力データの暗号化やハッシュ処理や拡散処理を行なう。この実施例１におけるデータ拡散例について、図１４を参照して説明する。

【０１０８】
図１４において、入力データの初期状態の長方形ステート２００中の左上端のバイト単位データ２０１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ２０１の構成ビットによって変化するバイト単位データを黒で示している。

【０１０９】
図から理解されるように、ラウンド演算の２ラウンド実行後には、左半分の分割データ全体のバイト単位データと、右半分の分割データの４つのバイト単位データの計２０バイトのデータに対する影響が発生し、３ラウンドの終了時には長方形ステート２００中、３２バイトすべてのデータに影響が発生している。なお、図では、長方形ステート２００中の左上端のバイト単位データ２０１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１１０】
次に、図１５を参照して、同様の処理における長方形ステート２００中の右半分の分割データの左上のバイト単位データ２０２、すなわち、最上位の行の左から５番目のデータについての影響について説明する。

【０１１１】
このアルゴリズムでは、第１ラウンド目の処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されるため、右側にあるデータは左側に移動する。いったん左側に移動すれば、先に図１４を参照して説明した長方形ステートの左上端のバイト単位データ２０１に対する処理と同様の処理が行なわれることになる。従って、その後の３ラウンド後にはすべてのデータに影響が発生する。従ってこのバイト単位データ２０２についても、全体で４ラウンドあれば、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１１２】
このように、本実施例では、
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（３）長方形ステートの半分の分割データに対する線形変換処理を行い、残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）

（４）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（５）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（５）の一連の処理、すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行することで効率的な攪拌性能を達成している。

【０１１３】

本実施例における演算コストについて考察する。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかった演算コストを見積もる。図１４、図１５を参照して説明したように、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに最大４ラウンドを要する。

【０１１４】

従って、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに、最大４回ずつの非線形変換処理（Ｈ－ＳＵＢ）、シフト処理（Ｈ－ＳＨＩＦＴ）、線形変換および排他的論理和処理（ＭＡＴ－ＸＯＲ）、スワップ処理（ＳＷＡＰ）、鍵適用演算処理（Ｈ－ＫＡＤＤ）が必要である。

【０１１５】

先に説明したように、演算処理を実行するためには、論理回路や演算処理プログラムなどが利用され、その構成によって必要とする演算回路や処理速度も異なることになる。従って絶対的な効率の評価は難しいが、上記の演算に必要となる論理回路におけるゲート数を１つの評価指標とすることが可能である。

例えば、先に説明したように、１２８ビットの正方ステートに対して設定される論理回路実装例として、各演算に必要とするゲート数は、

ＳＵＢ演算＝３，２００～４，８００ゲート程度、

ＭＡＴ演算＝８００～１，２００ゲート程度、

ＫＡＤＤ演算＝３２０ゲート程度、

これらのゲート数に相当する。

【０１１６】

２５６ビットの長方形ステートに対する処理における演算のコストは以下のように換算することができる。

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）は、上記１２８ビットの正方ステートに対する非線形変換処理（ＳＵＢ）と同等、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）はゲートを通過する必要がなく演算コスト＝０、

（３）長方形ステートの半分の分割データに対する線形変換処理を行い、残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）は、１２８ビットの正方ステートに対する線形変換処理（ＭＡＴ）と鍵適用演算処理（ＫＡＤＤ）を併せた処理と同等、

（４）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）はゲートを通過する必要がなく演算コスト＝０、

（５）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）は、１２８ビットの正方ステートに対する鍵適用演算処理（ＫＡＤＤ）と同等、

このように推定される。

【０１１７】

従って、本実施例において、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、

演算コスト＝４ＳＵＢ＋４ＭＡＴ＋８ＫＡＤＤ

と見積もることができる。
上記ゲート数、すなわち、
ＳＵＢ演算＝３，２００～４，８００ゲート程度、
ＭＡＴ演算＝８００～１，２００ゲート程度、
ＫＡＤＤ演算＝３２０ゲート程度、
これらの演算対応ゲート数に基づいて、演算コストを算出すると、
演算コスト＝４ＳＵＢ＋４ＭＡＴ＋８ＫＡＤＤ
＝１９Ｋゲート～２７Ｋゲート
となる。

【０１１８】

これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］における変換処理において必要としていた演算コスト（２６Ｋゲート～３８Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化される。具体的には処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０１１９】

［実施例２］

次に、本発明の第２実施例におけるラウンド演算の構成を図１６に示す。実施例２のラウンド演算の処理シーケンスは、以下の通りである。

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）
（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）
（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの（１）～（６）の一連の処理をラウンド演算とする。

【０１２０】

この実施例２のラウンド演算処理を要約すると、データ変換装置のデータ変換部は以下のラウンド演算を実行することになる。長方形ステートを２分割した分割データの一方の分割データＡに対して非線形変換処理と、シフト処理を実行し、さらに線形変換処理を実行して分割データＡの更新を行い、さらに、更新された分割データＡと、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行する。このようなラウンド処理を実行する。

【０１２１】

なお、実施例においては長方形ステートの左半分の分割データを選択して非線形変換処理（Ｈ－ＳＵＢ）などの主要演算処理を実行する構成として説明するが、主要演算を実行する対象は右半分とする構成としてもよい。すなわち、以下で説明する処理において、左右のデータを入れ替えた処理を行なっても効果は同様である。

【０１２２】

各処理の詳細について説明する。
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
これらの処理は、実施例１において説明した処理と同様の処理である。

【０１２３】

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）は、

図８（１）に示すように、変換処理後のバイト単位の出力b_iと入力a_iの関係は、
$b_i = S(a_i)$，if＝i＝１～４，９～１２，１７～２０，２５～２８，
$b_i = a_i$，else
上記のように右半分の分割データは変更しない。

【０１２４】

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）は、図１３（Ａ）を参照して説明したように、各行について異なるシフト量（０～（ｎ－１））のシフトを実行する。例えば、先に図２を参照して説明した正方ステートに対するシフト処理と同様、一行目ローテーションシフトなし、二行目は１バイト分右方向へローテーションシフト、三行目は２バイト分、四行目は３バイト分のローテーションシフトを行う。

【０１２５】

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

この処理は、先に図８（２）を参照して説明した処理と同様の処理であり、長方形ステートの左半分の各バイトデータについて、列ごとの４つのデータをベクトルとみなし４×４の行列［Ｍ］による演算を施して値を更新する演算である。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$${}^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = M\,{}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$$

i＝１，２，３，４、

$${}^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = {}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$$

i＝５，６，７，８、

上記のように、長方形ステートの左半分の各バイトデータについて行列［Ｍ］を適用した線形変換を実行し、右半分の分割データは変更しない。

なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【０１２６】

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）

この排他的論理和演算（ＸＯＲ）処理について、図１７を参照して説明する。長方形ステートの左半分の１つの列データと、右半分の対応するデータ列と排他的論理和（ＸＯＲ）を行い、その結果を右半分の分割データとして更新する。左半分の分割データは更新しない。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

（３－１）左半分の分割データ

$${}^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = {}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$$

i＝１，２，３，４、

（３－２）右半分の分割データ

$${}^t(b_{i+4}, b_{i+12}, b_{i+20}, b_{i+28}) = {}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})\ (\mathrm{XOR})\ {}^t(a_{i+4}, a_{i+12}, a_{i+20}, a_{i+28})$$

i＝１，２，３，４、

である。なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【０１２７】

すなわち、左半分の分割データの左から第１列データと、右半分の分割データの左からの第１列データ（トータルでは左から５番目）との排他的論理和（ＸＯＲ）を実行して、その結果を右半分の左からの第１列データ（トータルでは左から５番目）の更新データとする。以下、同様に、左半分の分割データの左から第２列データと、右半分の分割データの左からの第２列データ（トータルでは左から６番目）との排他的論理和（ＸＯＲ）を実行して、その結果を右半分の左から第２列データ（トータルでは左から６番目）の更新デ

ータとする。以下同様の処理を行なう。

【０１２８】

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

このスワップ処理（ＳＷＡＰ）は、先に説明した実施例１の処理と同様の処理であり、図１３（Ｃ）に示すように、長方形ステートにおける左半分と右半分を入れ替える処理である。

【０１２９】

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

この処理は、先に図８（３）を参照して説明した処理と同様の処理であり、長方形ステートの左半分の各バイトデータについて、鍵スケジュール部から出力されたラウンド鍵［k_i］を各バイトデータに排他的論理和を行う。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

$b_i = a_i$（ＸＯＲ）k_i，ｉｆ　ｉ＝１～４，９～１２，１７～２０，２５～２８，

$b_i = a_i$，ｅｌｓｅ

上記のように右半分の分割データは変更しない。

である。なお、上記式において（ＸＯＲ）は排他的論理和演算を示している。

【０１３０】

本発明の第２実施例におけるラウンド演算は、図１６に示すように、

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（６）の一連の処理によって構成される。すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行する。

【０１３１】

このラウンド演算を繰り返し実行することで、入力データの暗号化やハッシュ処理や拡散処理を行なう。この実施例２におけるデータ拡散例について、図１８を参照して説明する。

【０１３２】

図１８において、入力データの初期状態の長方形ステート２２０中の左上端のバイト単位データ２２１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ２２１の構成ビットによって変化するバイト単位データを黒で示している。

【０１３３】

図から理解されるように、ラウンド演算の２ラウンド終了時には、長方形ステートの全てのバイト単位データに影響が発生している。なお、図では、長方形ステート２２０中の左上端のバイト単位データ２２１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１３４】

次に、図１９を参照して、本実施例２の処理における長方形ステート２２０中の右半分の分割データの左上のバイト単位データ２２２、すなわち、最上位の行の左から５番目のデータについての影響について説明する。

【０１３５】

このアルゴリズムでは、先に説明した実施例１と同様、第１ラウンド目の処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されるため、右側にあるデータは左側に移動する。いったん左側に移動すれば、先に図１８を参照して説明した長方形ステートの左上端のバイト単位データ２２１に対する処理と同様の処理が行なわれることになる。従って、その後の２ラウンド後にはすべてのデータに影響が発生する。従ってこのバイト単位データ２２２についても、全体で３ラウンドあれば、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。このように実施例２は、実施例１よりもさらに効率的な攪拌性能を達成している。

【０１３６】

このように、本実施例では、

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（６）の一連の処理、すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順に従った処理を１つのラウンド演算処理として実行することで効率的な攪拌性能を達成している。

【０１３７】

本実施例における演算コストについて考察する。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかった演算コストを見積もる。図１８、図１９を参照して説明したように、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに最大３ラウンドを要する。

【０１３８】

従って、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに、最大３回ずつの非線形変換処理（Ｈ－ＳＵＢ）、シフト処理（Ｈ－ＳＨＩＦＴ）、線形変換処理（Ｈ－ＭＡＴ）、排他的論理和処理（ＸＯＲ）、スワップ処理（ＳＷＡＰ）、鍵適用演算処理（Ｈ－ＫＡＤＤ）が必要である。

【０１３９】

２５６ビットの長方形ステートに対する処理における演算のコストは以下のように換算できる。

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）は、上記１２８ビットの正方ステートに対する非線形変換処理（ＳＵＢ）と同等、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）はゲートを通過する必要がなく演算コスト＝０、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）は、１２８ビットの正方ステートに対する線形変換処理（ＭＡＴ）と同等、

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）は１２８ビットの正方ステートに対する鍵適用演算処理（ＫＡＤＤ）と同等、

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）はゲートを通過する必要がなく演算コスト＝０、

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）は、１２８ビットの正方ステートに対する鍵適用演算処理（ＫＡＤＤ）と同等、

このように推定される。

【０１４０】

従って、本実施例において、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、

演算コスト＝３ＳＵＢ＋３ＭＡＴ＋６ＫＡＤＤ

と見積もることができる。

先に説明したゲート数、すなわち、

ＳＵＢ演算＝３，２００～４，８００ゲート程度、

ＭＡＴ演算＝８００～１，２００ゲート程度、

ＫＡＤＤ演算＝３２０ゲート程度、

これらの演算対応ゲート数に基づいて、演算コストを算出すると、

演算コスト＝３ＳＵＢ＋３ＭＡＴ＋６ＫＡＤＤ

＝１４Ｋゲート～２０Ｋゲート

となる。

【０１４１】

これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］における変換処理において必要としていた演算コスト（２６Ｋゲート～３８Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化される。具体的には処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０１４２】

［実施例３］

次に、本発明の第３実施例におけるラウンド演算の構成を図２０に示す。実施例３のラウンド演算の処理シーケンスは、以下の通りである。

（１）長方形ステートの半分の分割データの列ごとのデータに非線形変換（ＳＵＢ）を施した結果をベクトルとみなし線形変換行列［Ｍ］を適用して４×４の行列演算を施して線形変換を実行して、その結果を右半分の対応する列のデータと排他的論理和（ＸＯＲ）を行い、その結果で右半分の分割データを更新する非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）、

（２）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（３）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（４）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（４）の一連の処理をラウンド演算とする。

【０１４３】

この実施例３のラウンド演算処理を要約すると、データ変換装置のデータ変換部は以下のラウンド演算を実行することになる。長方形ステートを２分割した分割データの一方の分割データＡに対して非線形変換処理と、線形変換処理を実行して他方の分割データＢとの排他的論理和を実行して、その結果を分割データＢの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対するシフト処理と鍵データとの排他的論理和処理を実行する。このようなラウンド処理を実行する。

【０１４４】

なお、実施例においては長方形ステートの左半分の分割データを選択して非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）などの主要演算処理を実行する構成として説明するが、主要演算を実行する対象は右半分とする構成としてもよい。すなわち、以下で説明する処理において、左右のデータを入れ替えた処理を行なっても効果は同様である。

【０１４５】

上記処理中、
（２）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（３）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（４）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの処理は、先に説明した実施例１，２の処理と同様であるので説明を省略する。

【０１４６】
（１）非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）について、図２１を参照して説明する。
長方形ステートの半分の分割データの列ごとのデータに非線形変換（ＳＵＢ）を施した結果をベクトルとみなし線形変換行列［Ｍ］を適用して４×４の行列演算を施して線形変換を実行して、その結果を右半分の対応する列のデータと排他的論理和（ＸＯＲ）を行い、その結果で右半分の分割データを更新する。左半分の分割データは更新しない。
変換処理後のバイト単位の出力b_iと入力a_iの関係は、
（１－１）左半分の分割データ
$^t(b_i, b_{i+8}, b_{i+16}, b_{i+24}) = {}^t(a_i, a_{i+8}, a_{i+16}, a_{i+24})$
ｉ＝１，２，３，４、
（１－２）右半分の分割データ
$^t(b_{i+4}, b_{i+12}, b_{i+20}, b_{i+28}) = [M\,{}^t(S(a_i), S(a_{i+8}), S(a_{i+16}), S(a_{i+24}))]\ (\mathrm{XOR})\ {}^t(a_{i+4}, a_{i+12}, a_{i+20}, a_{i+28})$
ｉ＝１，２，３，４、
である。なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【０１４７】
すなわち、左半分の分割データの左から第１列データに非線形変換（ＳＵＢ）を施した後、予め設定した線形変換行列［Ｍ］を適用して４×４の行列演算を施して、その結果と、右半分の分割データの左からの第１列データ（トータルでは左から５番目）との排他的論理和（ＸＯＲ）を実行した結果を、右半分の左からの第１列データ（トータルでは左から５番目）として更新する。以下、同様に、左半分の分割データの左から第２列データに非線形変換（ＳＵＢ）を施した後、予め設定した線形変換行列［Ｍ］を適用して４×４の行列演算を施して、その結果と、右半分の分割データの左からの第２列データ（トータルでは左から６番目）との排他的論理和（ＸＯＲ）を実行して、その結果を右半分の左からの第２列データ（トータルでは左から６番目）の更新データとする。以下同様の処理を行なう。

【０１４８】
本発明の第３実施例におけるラウンド演算は、図２０に示すように、
（１）長方形ステートの半分の分割データの列ごとのデータに非線形変換（ＳＵＢ）を施した結果をベクトルとみなし線形変換行列［Ｍ］を適用して４×４の行列演算を施して線形変換を実行して、その結果を右半分の対応する列のデータと排他的論理和（ＸＯＲ）を行い、その結果で右半分の分割データを更新する非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）、
（２）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（３）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（４）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの（１）～（４）の一連の処理によって構成される。すなわち、［ＳＵＢ－ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＫＡＤＤ］の順の処理を１つ

のラウンド演算処理として実行する。

【０１４９】

このラウンド演算を繰り返し実行することで、入力データの暗号化やハッシュ処理や拡散処理を行なう。この実施例３におけるデータ拡散例について、図２２を参照して説明する。

【０１５０】

図２２において、入力データの初期状態の長方形ステート２３０中の左上端のバイト単位データ２３１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ２３１の構成ビットによって変化するバイト単位データを黒で示している。

【０１５１】

図から理解されるように、ラウンド演算の２ラウンドの処理終了後には長方形ステートの左半分の全てのバイト単位データと、右半分の４つのバイト単位データ、計２０バイトのデータに影響が発生し、３ラウンドの終了時には長方形ステートの全てのバイト単位データに影響が発生している。なお、図では、長方形ステート２３０中の左上端のバイト単位データ２３１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１５２】

次に、図２３を参照して、本実施例３の処理における長方形ステート２３０中の右半分の分割データの左上のバイト単位データ２３２、すなわち、最上位の行の左から５番目のデータについての影響について説明する。

【０１５３】

このアルゴリズムでは、先に説明した実施例１，２と同様、第１ラウンド目の処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されるため、右側にあるデータは左側に移動する。いったん左側に移動すれば、先に図２２を参照して説明した長方形ステートの左上端のバイト単位データ２３１に対する処理と同様の処理が行なわれることになる。従って、その後の３ラウンド後にはすべてのデータに影響が発生する。従ってこのバイト単位データ２３２についても、全体で４ラウンドあれば、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１５４】

さらにこの実施例３の方式の場合、非線形変換処理（ＳＵＢ）や行列［Ｍ］を適用した線形変換処理（ＭＡＴ）のような個別の演算処理における入力情報が出力情報によって上書きする処理を伴わない。すなわち、図２０に示すラウンド演算処理の（１）の非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）では、分割データＡに対して非線形変換処理と、シフト処理と線形変換処理を実行するが、この結果データによる分割データＡの更新は行われない。

【０１５５】

従って、例えば、図２０に示すラウンド演算を繰り返して実行して暗号結果を生成し、その暗号結果を復号するアルゴリズムにおいて、非線形変換や行列演算の逆演算を用いることなくデータを復元することが可能となる。従って、データ置換処理全体の逆関数を実装する際にも非線形変換処理（ＳＵＢ）や行列［Ｍ］を適用した線形変換処理（ＭＡＴ）の逆関数を実装することが不要となり、実装上、装置の小型化やコスト安をもたらす効果がある。このように本実施例３は、先に説明した実施例１と同等の攪拌性能を保持するとともに優れた実装上の特徴を有している。

【０１５６】

本実施例では、

（１）長方形ステートの半分の分割データの列ごとのデータに非線形変換（ＳＵＢ）を施した結果をベクトルとみなし線形変換行列［Ｍ］を適用して４×４の行列演算を施して線形変換を実行して、その結果を右半分の対応する列のデータと排他的論理和（ＸＯＲ）を行い、その結果で右半分の分割データを更新する非線形変換＆線形変換＆排他的論理和

処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）、
（２）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（３）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（４）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの（１）～（４）の一連の処理、すなわち、［ＳＵＢ－ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＫＡＤＤ］の順に従った処理を１つのラウンド演算処理として実行することで効率的な攪拌性能を達成している。

【０１５７】
本実施例における演算コストについて考察する。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかった演算コストを見積もる。図２２、図２３を参照して説明したように、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに最大４ラウンドを要する。

【０１５８】
従って、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに、最大４回ずつの非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）、スワップ処理（ＳＷＡＰ）、シフト処理（Ｈ－ＳＨＩＦＴ）、鍵適用演算処理（Ｈ－ＫＡＤＤ）が必要である。

【０１５９】
２５６ビットの長方形ステートに対する処理における演算のコストは以下のように換算できる。
（１）非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）は、１２８ビットの正方ステートに対する非線形変換処理（ＳＵＢ）と線形変換処理（ＭＡＴ）と鍵適用演算処理（ＫＡＤＤ）を全て行なう演算コストに相当する。
（２）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）はゲートを通過する必要がなく演算コスト＝０、
（３）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）はゲートを通過する必要がなく演算コスト＝０、
（４）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）は１２８ビットの正方ステートに対する鍵適用演算処理（ＫＡＤＤ）と同等、
このように推定される。

【０１６０】
従って、本実施例において、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、
演算コスト＝４ＳＵＢ＋４ＭＡＴ＋８ＫＡＤＤ
と見積もることができる。
先に説明したゲート数、すなわち、
ＳＵＢ演算＝３，２００～４，８００ゲート程度、
ＭＡＴ演算＝８００～１，２００ゲート程度、
ＫＡＤＤ演算＝３２０ゲート程度、
これらの演算対応ゲート数に基づいて、演算コストを算出すると、
演算コスト＝４ＳＵＢ＋４ＭＡＴ＋８ＫＡＤＤ
＝１９Ｋゲート～２７Ｋゲート
となる。

【０１６１】
これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］における変換処理において必要としていた演算コスト（２６Ｋゲート～３８Ｋゲート

）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化される。具体的には処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０１６２】

［実施例４］

次に、本発明の第４実施例におけるラウンド演算の構成を図２４に示す。実施例４のラウンド演算の処理シーケンスは、以下の通りである。

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ２）

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（６）の一連の処理をラウンド演算とする。

【０１６３】

この実施例４のラウンド演算処理を要約すると、データ変換装置のデータ変換部は以下のラウンド演算を実行することになる。長方形ステートを２分割した分割データの一方の分割データＡに対して非線形変換処理と、シフト処理と線形変換処理を実行し、さらに、他方の分割データＢとの排他的論理和を実行して、その結果を分割データＡの更新データとして設定し、さらに分割データＡＢのスワップ処理の後、分割データＡに対する鍵データとの排他的論理和処理を実行する。このようなラウンド処理を実行する。

【０１６４】

なお、実施例においては長方形ステートの左半分の分割データを選択して非線形変換処理（Ｈ－ＳＵＢ）などの主要演算処理を実行する構成として説明するが、主要演算を実行する対象は右半分とする構成としてもよい。すなわち、以下で説明する処理において、左右のデータを入れ替えた処理を行なっても効果は同様である。

【０１６５】

この実施例４のアルゴリズムは、先に図１６を参照して説明した実施例２のアルゴリズムにおける（１）～（６）の処理中、（１）非線形変換処理（Ｈ－ＳＵＢ）、（２）シフト処理（Ｈ－ＳＨＩＦＴ）、（３）線形変換処理（Ｈ－ＭＡＴ）と、（５）スワップ処理（ＳＷＡＰ）、（６）鍵適用処理（Ｈ－ＫＡＤＤ）の各処理は同じであり、（４）の処理のみが異なる。

【０１６６】

実施例２における（４）排他的論理和演算（ＸＯＲ）では、長方形ステートの右半分の分割データを更新し、左半分の分割データは更新しないという処理であったが、本実施例４では、長方形ステートの右半分の分割データは更新せず、左半分の分割データを更新する処理を実行する。

【０１６７】

本実施例における（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ２）について、図２５を参照して説明する。

【０１６８】

長方形ステートの左半分の１つの列データと、右半分の対応するデータ列と排他的論理和（ＸＯＲ）を行い、その結果を左半分の分割データとして更新する。右半分の分割データは更新しない。

変換処理後のバイト単位の出力b_iと入力a_iの関係は、

（３－１）左半分の分割データ

$^{t}(b_{i}, b_{i+8}, b_{i+16}, b_{i+24}) = {}^{t}(a_{i}, a_{i+8}, a_{i+16}, a_{i+24})$（ＸＯＲ）$^{t}(a_{i+4}, a_{i+12}, a_{i+20}, a_{i+28})$

ｉ＝１，２，３，４、

（３－２）右半分の分割データ

$^{t}(b_{i+4}, b_{i+12}, b_{i+20}, b_{i+28}) = {}^{t}(a_{i}, a_{i+12}, a_{i+20}, a_{i+28})$

ｉ＝１，２，３，４、

である。なお、^{t}（）は、行列における行と列を入れ替えた転置行列を示している。

【０１６９】

すなわち、左半分の分割データの左から第１列データと、右半分の分割データの左からの第１列データ（トータルでは左から５番目）との排他的論理和（ＸＯＲ）を実行して、その結果を左半分の左から第１列データの更新データとする。以下、同様に、左半分の分割データの左から第２列データと、右半分の分割データの左から第２列データ（トータルでは左から６番目）との排他的論理和（ＸＯＲ）を実行して、その結果を左半分の左からの第２列データの更新データとする。以下同様の処理を行なう。

【０１７０】

本発明の第４実施例におけるラウンド演算は、図２４に示すように、

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ２）

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_{i}］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（６）の一連の処理によって構成される。すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ２］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行する。

【０１７１】

このラウンド演算を繰り返し実行することで、入力データの暗号化やハッシュ処理や拡散処理を行なう。この実施例４におけるデータ拡散例について、図２６を参照して説明する。

【０１７２】

図２６において、入力データの初期状態の長方形ステート２４０中の左上端のバイト単位データ２４１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ２４１の構成ビットによって変化するバイト単位データを黒で示している。

【０１７３】

図から理解されるように、ラウンド演算の４ラウンド終了時には、長方形ステートの全てのバイト単位データに影響が発生している。なお、図では、長方形ステート２４０中の左上端のバイト単位データ２４の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１７４】

また、右半分に含まれるデータの影響については、先に説明した実施例１～３と同様、１つのラウンド処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されることになり、左半分の分割データと右半分の分割データの入れ替えが実行され、右側データが、いったん左側に移動すれば、後の処理は、図２６に示すシーケンスで実行され、結果として、４＋１＝５ラウンド終了時には、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０１７５】
この実施例４の特徴としては、あるラウンドに入力される半分の分割データは、前のラウンドで更新されることがないため、適切にデータを保存しておけば、前のラウンドの処理が終わるのを待つことなく、次のラウンドの処理をスタートさせることが可能である点である。この処理によって高速な処理が可能となる。具体的には排他的論理和［ＸＯＲ２］の直前までは並列して処理することが可能な構成である。
【０１７６】
このように、本実施例では、
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）
（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ２）
（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの（１）～（６）の一連の処理、すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ２］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順に従った処理を１つのラウンド演算処理として実行することで効率的な攪拌性能を達成している。
【０１７７】
本実施例における演算コストについて考察する。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかった演算コストを見積もる。図２６を参照して説明したように、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに最大５ラウンドを要する。
【０１７８】
従って、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでに、最大５回ずつの非線形変換処理（Ｈ－ＳＵＢ）、シフト処理（Ｈ－ＳＨＩＦＴ）、線形変換処理（Ｈ－ＭＡＴ）、排他的論理和処理（ＸＯＲ２）、スワップ処理（ＳＷＡＰ）、鍵適用演算処理（Ｈ－ＫＡＤＤ）が必要である。
【０１７９】
２５６ビットの長方形ステートに対する処理における演算のコストは以下のように換算できる。
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）は、上記１２８ビットの正方ステートに対する非線形変換処理（ＳＵＢ）と同等、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）はゲートを通過する必要がなく演算コスト＝０、
（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）は、１２８ビットの正方ステートに対する線形変換処理（ＭＡＴ）と同等、
（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）は１２８ビットの正方ステートに対する鍵適用演算処理（ＫＡＤＤ）と同等、
（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）はゲートを通過する必要がなく演算コスト＝０、
（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）は、１２８ビットの正方ステートに対する鍵適用演算処理（ＫＡＤＤ）と同等、
このように推定される。

【０１８０】
従って、本実施例において、長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、
演算コスト＝５ＳＵＢ＋５ＭＡＴ＋１０ＫＡＤＤ
と見積もることができる。
先に説明したゲート数、すなわち、
ＳＵＢ演算＝３，２００～４，８００ゲート程度、
ＭＡＴ演算＝８００～１，２００ゲート程度、
ＫＡＤＤ演算＝３２０ゲート程度、
これらの演算対応ゲート数に基づいて、演算コストを算出すると、
演算コスト＝５ＳＵＢ＋５ＭＡＴ＋１０ＫＡＤＤ
＝２３Ｋゲート～３３Ｋゲート
となる。
【０１８１】
これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］における変換処理において必要としていた演算コスト（２６Ｋゲート～３８Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化される。具体的には処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。
【０１８２】
［シフト処理（ＳＨＩＦＴ）の一般化構成について］
上述した実施例１～４においては長方形ステートにおける左半分の分割データにのみシフト処理を適用する［Ｈ－ＳＨＩＦＴ］を実行する構成例について説明した。すでに説明したように、シフト処理（ＳＨＩＦＴ）は各行ごとに異なる値のローテーションシフトを適用するものであるが、本発明の実施例１～４において実行するシフト処理（Ｈ－ＳＨＩＦＴ）に必要な性質は以下の通りである。
【０１８３】
＊同じ列に含まれるバイトデータはシフト演算後には必ず異なる列に含まれること
上記性質が満たされていれば、必ずしもローテーションに基づくシフトである必要はない。
【０１８４】
さらに、各実施例において説明したシフト処理（Ｈ－ＳＨＩＦＴ）は、長方形ステートの２つの半分の分割データの両者に対して実行する構成としても、上述した各実施例において説明した効果をもたらす。先に説明したようにシフト処理事態は演算コスト＝０と解釈できるので、処理対象が増加しても演算ストを増加させることにはならない。
【０１８５】
図２７は、先に説明した実施例１におけるシフト処理（Ｈ－ＳＨＩＦＴ）を長方形ステートの左右半分の分割データの両者に適用する処理のアルゴリズムである。長方形ステートの一方の半分の分割データを［ＳＬ］として、他方の半分の分割データを［ＳＲ］とした場合、ラウンド演算のアルゴリズムは、以下のようになる。
（１）長方形ステートの半分の分割データ［ＳＬ］に対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データ［ＳＬ］と［ＳＲ］に対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（３）長方形ステートの半分の分割データ［ＳＬ］に対する線形変換処理を行い、残り半分の分割データ［ＳＲ］との排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）
（４）長方形ステートの２つの半分の分割データ［ＳＬ］，［ＳＲ］を入れ替えるスワップ処理（ＳＷＡＰ）

（５）長方形ステートの半分の分割データ［ＳＬ］に対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

このような設定としても、先に説明した実施例１の拡散効果と同様の効果が得られる。他の実施例２，３，４についても同様であり、シフト処理は各実施例において説明した側の半分の分割データ以外の半分の分割データに対して適用してもよい。なお、左右両方のシフト処理の態様は、左右同じものでも左右独立に選んでも良い。

【０１８６】

［実施例１～４の一般化構成について］

上述した実施例１，２，３，４では、３２バイト（２５６ビット）データで構成される４行×８列、計３２個のバイト単位データによって構成される長方形ステートに対する処理例として説明をした。

【０１８７】

本発明は、この実施例に限らず様々な構成の長方形ステートに対して適用可能である。具体的には、左右に分割可能な偶数の列数（２ｎ）を有し、任意の行数（ｍ）を有する長方形ステート（$2n \neq m$）とした長方形ステートに対して適用することが可能である。

【０１８８】

例えば図２８に示すように全体で２ｍｎ個のバイト単位データを持つデータを、２ｎ列、ｍ行のa_1～a_{2mn}の２ｍｎ個のバイト単位データを有する長方形ステートに対して、上述した実施例１～４において説明した非線形変換（Ｈ－ＳＵＢ）など、各種の変換処理によって変換を行う。変換結果である長方形ステートに含まれるバイト単位データをb_1～b_{2mn}とした場合、各実施例において定義した各処理は図２９に示すように以下の式によって一般形として示すことができる。

【０１８９】

（１）非線形変換処理（Ｈ－ＳＵＢ）

$b_i = S(a_i)$　if　$1 \leqq i \bmod 2n \leqq n$

$b_i = a_i$　else

ただしＳ（）は非線形変換処理を示す。

【０１９０】

（２）線形変換および排他的論理和演算（ＭＡＴ－ＸＯＲ）

for　i＝1...n

$^t(b_i, b_{i+2n}, \cdots, b_{i+2(m-1)n}) = {}^t(a_i, a_{i+2n}, \cdots, a_{i+2(m-1)n})$

$^t(b_{i+n}, b_{i+3n}, \cdots, b_{i+(2m-1)n}) = [M\,{}^t(a_i, a_{i+2n}, \cdots, a_{i+2(m-1)n})]$（ＸＯＲ）$^t(a_{i+n}, a_{i+3n}, \cdots, a_{i+(2m-1)n})$

ただし、

^{t}（）は転置行列を示し、

［Ｍ］は線形変換行列、

（ＸＯＲ）は排他的論理和演算である。

【０１９１】

（３）線形変換処理（Ｈ－ＭＡＴ）

$^t(b_i, b_{i+2n}, \cdots, b_{i+2(m-1)n}) = M\,{}^t(a_i, a_{i+2n}, \cdots, a_{i+2(m-1)n})$　if　i＝1..n

$^t(b_i, b_{i+2n}, \cdots, b_{i+2(m-1)n}) = {}^t(a_i, a_{i+2n}, \cdots, a_{i+2(m-1)n})$　else

ただし、

^{t}（）は転置行列を示し、

［Ｍ］は線形変換行列である。

【０１９２】

（４）鍵適用演算（Ｈ－ＫＡＤＤ）

$b_i = a_i$（XOR）k_i　　if　$1 \leqq i \bmod 2n \leqq n$
$b_i = a_i$　　else
ただし、
（XOR）は排他的論理和演算、
k_iは鍵データである。

【0193】
（5）排他的論理和処理（XOR）
for　i＝1..n
$^t(b_i, b_{i+2n}, .., b_{i+2(m-1)n}) = {}^t(a_i, a_{i+2n}, .., a_{i+2(m-1)n})$
$^t(b_{i+n}, b_{i+3n}, .., b_{i+(2m-1)n}) = {}^t(a_i, a_{i+2n}, .., a_{i+2(m-1)n})$（XOR）$^t(a_{i+n}, a_{i+3n}, .., a_{i+(2m-1)n})$
ただし、
^{t}（）は転置行列を示し、
（XOR）は排他的論理和演算である。

【0194】
（6）非線形変換＆線形変換＆排他的論理和処理（SUB－MAT－XOR）
for　i＝1..n
$^t(b_i, b_{i+2n}, .., b_{i+2(m-1)n}) = {}^t(a_i, a_{i+2n}, .., a_{i+2(m-1)n})$
$^t(b_i+n, b_{i+3n}, .., b_{i+(2m-1)n}) = [M\,{}^t(S(a_i), S(a_{i+2n}), .., S(a_{i+2(m-1)n})]$（XOR）$^t(a_{i+n}, a_{i+3n}, .., a_{i+(2m-1)n})$
ただし、
^{t}（）は転置行列を示し、
［M］は線形変換行列、
（XOR）は排他的論理和演算である。

【0195】
このように、上記実施例において説明した各処理は、上述のように、2n列、m行のa_1～a_{2mn}の2mn個のバイト単位データからなる長方形ステートに対する演算における入力［a_1～a_{2mn}］と出力［b_1～b_{2mn}］の関係式として上記のように一般形として定義できる。

【0196】
なお、シフト処理（H－SHIFT）に関しては、上記実施例では、長方形ステートの半分の分割データが正方ステートとなる例として説明していたが、任意の形状を持つ長方形ステートとした場合、半分の分割データが正方形ステートになるとは限らない。従って、このような場合も含めて、シフト処理（H－SHIFT）の一般形の定義について説明する。

【0197】
m行2n列の長方形ステートにおいて、シフト処理（H－SHIFT）対象となる半分の分割データ（m行n列）に対するシフト処理（H－SHIFT）に求められる処理ルールとして、以下のルールがある。
＊m≦nの場合：m行2n列の長方形ステートにおいて、シフト処理（H－SHIFT）対象となる半分の分割データ（m行n列）中の同じ列に含まれるバイトデータはシフト処理後には必ず異なる列に設定する。
＊m＞nの場合：シフト処理（H－SHIFT）対象となる半分の分割データ（m行n列）中の同じ列に含まれるバイトデータはシフト処理後の任意の列に（m／n）－1個以上、（m／n）＋1個以下の範囲内で含まれていること

【0198】

すなわち、シフト処理の実行に際して、ｍ行２ｎ列の長方形配列データ中、シフト処理対象となるｍ行ｎ列の分割データが、
ｍ≦ｎである場合、シフト前に同じ列のデータブロックがシフト処理後に異なる列になるようにシフトし、
ｍ＞ｎの場合には、シフト前に同じ列のデータブロックがシフト処理後の任意の列に（ｍ／ｎ）－１個以上、（ｍ／ｎ）＋１個以下の範囲内で含まれるようにシフト処理を実行する。

【０１９９】

ｍ行２ｎ列の長方形ステートにおいて、シフト処理（Ｈ－ＳＨＩＦＴ）対象となる半分の分割データ（ｍ行ｎ列）に対するシフト処理（Ｈ－ＳＨＩＦＴ）は、上記のルールに従って行う。このルールに従った処理を行うことで、ある列のデータが複数の列に効率的に拡散することが保証される。

【０２００】

［実施例１～４の一般化］

先に説明した実施例１～４では、長方形ステートの半分の分割データが正方ステートとなる例を説明したが、上述したように、本発明は、このような特殊な形状の長方形ステートのみならず、少なくとも２分割可能な偶数列を持つ任意の形状の長方形ステートに対して適用可能である。以下、長方形ステートの半分の分割データが正方ステートとならない場合に各実施例１～４を適用した場合のデータ拡散例について説明する。

【０２０１】

（実施例１の一般化例）

図３０は、長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステート３１０に対して、先に実施例１で説明したラウンド演算を実行した場合のデータ拡散例を示す図である。４行１２列の４８バイトの長方形ステート３１０中の左上端のバイト単位データ３１１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ３１１の構成ビットによって変化するバイト単位データを黒で示している。

【０２０２】

実施例１のラウンド演算アルゴリズムは、
（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、
（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、
（３）長方形ステートの半分の分割データに対する線形変換処理を行い、残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＭＡＴ－ＸＯＲ）
（４）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）
（５）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）
これらの（１）～（５）の一連の処理、すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行する処理である。

【０２０３】

図から理解されるように、ラウンド演算の４ラウンド終了時には長方形ステート３１０中、４８バイトすべてのデータに影響が発生している。なお、図では、長方形ステート３１０中の左上端のバイト単位データ３１１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２０４】

また、右半分に含まれるデータの影響については、先に説明した実施例で説明したと同様、１つのラウンド処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されることになり、左半分の分割データと右半分の分割データの入れ替えが実行され、右側データが、いったん左側に移動すれば、後の処理は、図３０に示すシーケンスで実行され、結果として、

４＋１＝５ラウンド終了時には、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２０５】

結果として、最大５ラウンドで長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えることが可能となる。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、

演算コスト＝７．５ＳＵＢ＋７．５ＭＡＴ＋１５ＫＡＤＤ

と見積もることができ、先に説明したゲート数換算で、演算コストを算出すると、

演算コスト＝７．５ＳＵＢ＋７．５ＭＡＴ＋１５ＫＡＤＤ

＝３５Ｋゲート～５０Ｋゲート

となる。これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］を４×１２のステートに拡張した方式では4ラウンドかかり、変換処理において必要とされる演算コストは１２ＳＵＢ＋１２ＭＡＴ＋１２ＫＡＤＤ（５２Ｋゲート～７６Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化され、処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０２０６】

（実施例２の一般化例）

図３１は、長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステート３２０に対して、先に実施例２で説明したラウンド演算を実行した場合のデータ拡散例を示す図である。４行１２列の４８バイトの長方形ステート３２０中の左上端のバイト単位データ３２１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ３２１の構成ビットによって変化するバイト単位データを黒で示している。

【０２０７】

実施例２のラウンド演算アルゴリズムは、

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ）

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（６）の一連の処理によって構成される。すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行する処理である。

【０２０８】

図３１から理解されるように、ラウンド演算の３ラウンド終了時には長方形ステート３２０中、４８バイトすべてのデータに影響が発生している。なお、図では、長方形ステート３２０中の左上端のバイト単位データ３２１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２０９】

また、右半分に含まれるデータの影響については、先に説明した実施例で説明したと同様、１つのラウンド処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されることになり、左半分の分割データと右半分の分割データの入れ替えが実行され、右側データが、いっ

たん左側に移動すれば、後の処理は、図３１に示すシーケンスで実行され、結果として、３＋１＝４ラウンド終了時には、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２１０】

結果として、最大４ラウンドで長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えることが可能となる。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、

演算コスト＝６ＳＵＢ＋６ＭＡＴ＋１２ＫＡＤＤ

と見積もることができ、先に説明したゲート数換算で、演算コストを算出すると、

演算コスト＝６ＳＵＢ＋６ＭＡＴ＋１２ＫＡＤＤ

＝２８Ｋゲート～４０Ｋゲート

となる。これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］を４×１２のステートに拡張した方式において必要としていた演算コスト（５２Ｋゲート～７６Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化され、処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０２１１】

（実施例３の一般化例）

図３２は、長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステート３３０に対して、先に実施例３で説明したラウンド演算を実行した場合のデータ拡散例を示す図である。４行１２列の４８バイトの長方形ステート３３０中の左上端のバイト単位データ３３１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ３３１の構成ビットによって変化するバイト単位データを黒で示している。

【０２１２】

実施例３のラウンド演算アルゴリズムは、

（１）長方形ステートの半分の分割データの列ごとのデータに非線形変換（ＳＵＢ）を施した結果をベクトルとみなし線形変換行列［Ｍ］を適用して４×４の行列演算を施して線形変換を実行して、その結果を右半分の対応する列のデータと排他的論理和（ＸＯＲ）を行い、その結果で右半分の分割データを更新する非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）、

（２）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（３）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（４）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（４）の一連の処理によって構成される。すなわち、［ＳＵＢ－ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＫＡＤＤ］の順の処理を１つのラウンド演算処理として実行する処理である。

【０２１３】

図３２から理解されるように、ラウンド演算の４ラウンド終了時には長方形ステート３３０中、４８バイトすべてのデータに影響が発生している。なお、図では、長方形ステート３３０中の左上端のバイト単位データ３３１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２１４】

また、右半分に含まれるデータの影響については、先に説明した実施例で説明したと同様、１つのラウンド処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されることになり

、左半分の分割データと右半分の分割データの入れ替えが実行され、右側データが、いったん左側に移動すれば、後の処理は、図３２に示すシーケンスで実行され、結果として、４＋１＝５ラウンド終了時には、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２１５】

結果として、最大５ラウンドで長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えることが可能となる。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、

演算コスト＝７．５ＳＵＢ＋７．５ＭＡＴ＋１５ＫＡＤＤ

と見積もることができ、先に説明したゲート数換算で、演算コストを算出すると、

演算コスト＝７．５ＳＵＢ＋７．５ＭＡＴ＋１５ＫＡＤＤ

＝３５Ｋゲート～５０Ｋゲート

となる。これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］を４×１２のステートに拡張した方式において必要としていた演算コスト（５２Ｋゲート～７６Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化され、処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０２１６】

（実施例４の一般化例）

図３３、図３４は、長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステート３４０に対して、先に実施例４で説明したラウンド演算を実行した場合のデータ拡散例を示す図である。４行１２列の４８バイトの長方形ステート３４０中の左上端のバイト単位データ３４１に着目する（黒でマーク）。各ラウンド演算の処理において、このバイト単位データ３４１の構成ビットによって変化するバイト単位データを黒で示している。

【０２１７】

実施例４のラウンド演算アルゴリズムは、

（１）長方形ステートの半分の分割データに対する非線形変換処理（Ｈ－ＳＵＢ）、

（２）長方形ステートの半分の分割データに対するシフト処理（Ｈ－ＳＨＩＦＴ）、

（３）長方形ステートの半分の分割データに対する線形変換処理（Ｈ－ＭＡＴ）

（４）長方形ステートの半分の分割データと残り半分の分割データとの排他的論理和演算（ＸＯＲ）を行なう処理（ＸＯＲ２）

（５）長方形ステートの２つの半分の分割データを入れ替えるスワップ処理（ＳＷＡＰ）

（６）長方形ステートの半分の分割データに対して、ラウンド鍵［k_i］を適用して排他的論理和（ＸＯＲ）を行う処理（Ｈ－ＫＡＤＤ）

これらの（１）～（６）の一連の処理、すなわち、［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ２］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］の順に従った処理を１つのラウンド演算処理として実行する処理である。

【０２１８】

図３３、図３４から理解されるように、ラウンド演算の６ラウンド終了時には長方形ステート３４０中、４８バイトすべてのデータに影響が発生している。なお、図では、長方形ステート３４０中の左上端のバイト単位データ３４１の影響の広がりを示しているが、この影響の広がりかたは左半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２１９】

また、右半分に含まれるデータの影響については、先に説明した実施例で説明したと同様、１つのラウンド処理内にスワップ［ＳＷＡＰ］演算が必ず一度実行されることになり

、左半分の分割データと右半分の分割データの入れ替えが実行され、右側データが、いったん左側に移動すれば、後の処理は、図３３、図３４に示すシーケンスで実行され、結果として、６＋１＝７ラウンド終了時には、影響が長方形全てのバイト単位データに影響する。このことは、右半分に含まれるすべてのバイトデータについて同様に言える性質である。

【０２２０】

結果として、最大７ラウンドで長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えることが可能となる。長方形ステートを構成する１つのバイト単位データが、長方形ステートを構成するすべてのバイトに影響を与えるまでにかかる演算コストは、

演算コスト＝１０．５ＳＵＢ＋１０．５ＭＡＴ＋２１ＫＡＤＤ

と見積もることができ、先に説明したゲート数換算で、演算コストを算出すると、

演算コスト＝１０．５ＳＵＢ＋１０・５ＭＡＴ＋２１ＫＡＤＤ

＝５０Ｋゲート～７０Ｋゲート

となる。

これは、先に、長方形ステートの処理例として説明したアルゴリズム［Ｒｉｊｎｄａｅｌ］を４×１２のステートに拡張した方式において必要としていた演算コスト（５２Ｋゲート～７６Ｋゲート）に比べて優位であり、本実施例に従った処理を実行することで、長方形ステートに対する拡散がより効率化され、処理速度の向上や、装置の小型化などを実現することができ、拡散性能の向上により暗号データのセキュリティレベルや秘匿性のレベルも増大させることが可能となる。

【０２２１】

［ラウンド演算における処理順序の入れ替えについて］

上述した各実施例では、ラウンド演算における処理シーケンスを１つのシーケンスとして説明している。すなわち、

実施例１は、

［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］

実施例２は、

［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］

実施例３は、

［ＳＵＢ－ＭＡＴ－ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＫＡＤＤ］

実施例４は、

［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ２］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］

このシーケンスで各ラウンド演算が実行されるものとして説明した。しかし、必ずしもこのシーケンスに限定されることなく、ラウンド演算において実行する処理順は異なる順番としてもよい。例えば、ラウンド内の初めの処理を別の処理に設定して、ラウンドの切れ目を変えて別のラウンド構成として解釈することは容易である。

【０２２２】

例えば実施例１や実施例２では、非線形変換［Ｈ－ＳＵＢ］とシフト処理［Ｈ－ＳＨＩＦＴ］は順序を入れ替えても拡散性能に影響はない、またいずれの実施例においても鍵適用演算［Ｈ－ＫＡＤＤ］の位置がどの位置に挿入されようとも拡散性能には影響がない。従って、本方式で説明した拡散性能が変わらない範囲で処理順序を入れ替えることは可能であり、そのような構成も本発明に含まれる。

【０２２３】

［ＤＳＭの適用］

上述した実施例において、線形変換処理［Ｈ－ＭＡＴ］に適用する行列は各ラウンドにおいて共通の固定行列［Ｍ］を利用することが可能であるが、この行列［Ｍ］をラウンド

間で切り換えて異なる行列を適用する構成としてもよい。いわゆるＤＳＭ（Ｄｉｆｆｕｓｉｏｎ　Ｓｗｉｔｃｈｉｎｇ　Ｍｅｃｈａｎｉｓｍ）の適用構成とすることでさらにセキュリティレベルを向上させることができる。

【０２２４】

実施例２においてＤＳＭを適用する場合の例について説明する。実施例２のラウンド演算は、先に図１６を参照して説明したように、

［Ｈ－ＳＵＢ］→［Ｈ－ＳＨＩＦＴ］→［Ｈ－ＭＡＴ］→［ＸＯＲ］→［ＳＷＡＰ］→［Ｈ－ＫＡＤＤ］

これらの複数の処理によって構成されるラウンド演算を繰り返し実行する処理である。

【０２２５】

あるラウンドにおいて線形変換処理［Ｈ－ＭＡＴ］によって、長方形ステートの左半分の分割データに含まれる各列に対して、線形変換行列［Ｍ］による行列演算が施された結果がその列に格納される。この線形変換処理［Ｈ－ＭＡＴ］によって更新された左半分の分割データは、その後のスワップ［ＳＷＡＰ］処理により右半分に移動する。

【０２２６】

次のラウンドの線形変換処理［Ｈ－ＭＡＴ］では、その時点で左半分に含まれる各列に対して、さらに線形変換行列［Ｍ］による行列演算が施された結果が格納され、さらに直後の排他的論理和処理［ＸＯＲ］により左半分の分割データが右半分の分割データに排他的論理和されるという構成を持つ。

【０２２７】

この排他的論理和処理［ＸＯＲ］が終わった直後の右半分に含まれる各列のデータに着目すると、各列は２回分の行列演算の結果が足し込まれた形になっていることがわかる。長方形ステートを構成する１つの列のベクトルを^{t}（Ｘ，Ｙ，Ｚ，Ｗ）で表すと、以下の一般式で書くことができる。

【数２】

$$\begin{pmatrix} W \\ X \\ Y \\ Z \end{pmatrix} = M\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \oplus M\begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix} = M\begin{pmatrix} a \oplus e \\ b \oplus f \\ c \oplus g \\ d \oplus h \end{pmatrix}$$

【０２２８】

ここでは（ａ，ｂ，ｃ，ｄ）を先行ラウンドの線形変換処理［Ｈ－ＭＡＴ］で行列に入力された列ベクトルの要素とし、（ｅ，ｆ，ｇ，ｈ）を次のラウンドの線形変換処理［Ｈ－ＭＡＴ］で行列に入力された列ベクトルの要素であるものとする。

【０２２９】

このとき、（ａ，ｂ，ｃ，ｄ，ｅ，ｆ，ｇ，ｈ）に含まれる非ゼロの要素数と、結果データである（Ｗ，Ｘ，Ｙ，Ｚ）の非ゼロの要素数の関係に着目する。たとえばａ＝ｅ≠０，ｂ＝ｃ＝ｄ＝ｆ＝ｇ＝ｈ＝０である場合に、Ｗ＝Ｘ＝Ｙ＝Ｚ＝０となることがわかる。２つの要素ａとｅが０ではないのにも関わらず、演算結果に非ゼロの要素が一つもないことになっており、入出力に含まれる非ゼロ要素の総和が２となる状態が発生してしまう。

【０２３０】

置換関数中の中ではあるデータがなるべく多くのデータに影響すると同時に、入出力に含まれる非ゼロ要素の総和ができるだけ低い水準にならないことも求められている。これは差分攻撃や線形攻撃に対しての対策となる。

【０２３１】
このような脆弱性に対する対策としては、全てのラウンドにおいて適用する線形変換行列［Ｍ］を１つの固定行列とするのではなく複数の異なる行列、例えば２つの行列［Ｍ１］，［Ｍ２］を利用する、いわゆるＤＳＭ（Ｄｉｆｆｕｓｉｏｎ Ｓｗｉｔｃｈｉｎｇ Ｍｅｃｈａｎｉｓｍ）を適用することが有効である。なお、ＤＳＭを適用した暗号アルゴリズムについては、例えば本出願人と同一の特許出願である特開２００７－１９９１５６などに記載されている。
【０２３２】
２つの行列Ｍ１，Ｍ２を準備し、以下に示す関係式を常に満たすように２つの行列を配置する。
【数３】

$$\begin{pmatrix} W \\ X \\ Y \\ Z \end{pmatrix} = M_1 \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \oplus M_2 \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

【０２３３】
または
【数４】

$$\begin{pmatrix} W \\ X \\ Y \\ Z \end{pmatrix} = M_2 \begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} \oplus M_1 \begin{pmatrix} e \\ f \\ g \\ h \end{pmatrix}$$

【０２３４】
上記のいずれかの関係を常に満たすように２つの行列［Ｍ１］，［Ｍ２］を配置する。
【０２３５】
これらの行列［Ｍ１］、［Ｍ２］を並べて得られる行列Ｍ１｜Ｍ２の分岐数が３以上、またはそれぞれの逆行列を転置して得られる行列を並べて得られる行列${}^{t}Ｍ１^{-1}$｜${}^{t}Ｍ２^{-1}$の分岐数が３以上になるように設定する。両者を同時に満たす構成にしてもよい。このようにすることで非ゼロの要素が少ないまま上記の状態が回避され、差分攻撃や線形攻撃に対する耐性を向上させる効果がある。また、実施例４においても同様な関係式が導けるため、上記のＤＳＭ構成を適用すれば同様の効果が期待できる。
【０２３６】

［３．データ変換装置の構成例］

最後に、上述した実施例に従った処理を実行する装置としてのＩＣモジュール７００の構成例を図３５に示す。上述の処理は、例えばＰＣ、ＩＣカード、リーダライタ、その他、様々な情報処理装置において実行可能である。また、各処理は、論理回路を構成したハード回路やプログラム、あるいはその双方を適用して実行可能である。処理を実行する一例として例えば、図３５に示すＩＣモジュール７００があり、このようなＩＣモジュール７００は様々な機器に搭載することが可能である。

【０２３７】

図３５に示すＣＰＵ(Central processing Unit)７０１は、暗号処理の開始や、終了、データの送受信の制御、各構成部間のデータ転送制御、その他の各種プログラムを実行するプロセッサである。メモリ７０２は、ＣＰＵ７０１が実行するプログラム、あるいは演算パラメータなどの固定データを格納するＲＯＭ（Read-Only-Memory）、ＣＰＵ７０１の処理において実行されるプログラム、およびプログラム処理において適宜変化するパラメータの格納エリア、ワーク領域として使用されるＲＡＭ（Random Access Memory）等からなる。また、メモリ７０２は暗号処理に必要な鍵データや、暗号処理において適用する変換テーブル（置換表）や変換行列に適用するデータ等の格納領域として使用可能である。なおデータ格納領域は、耐タンパ構造を持つメモリとして構成されることが好ましい。

【０２３８】

データ変換部７０３は、例えば上述した各種の暗号処理、

（１）非線形変換処理（Ｈ－ＳＵＢ）

（２）線形変換および排他的論理和演算（ＭＡＴ－ＸＯＲ）

（３）線形変換処理（Ｈ－ＭＡＴ）

（４）鍵適用演算（Ｈ－ＫＡＤＤ）

（５）排他的論理和処理（ＸＯＲ）（ＸＯＲ２）

（６）非線形変換＆線形変換＆排他的論理和処理（ＳＵＢ－ＭＡＴ－ＸＯＲ）

これらの各処理を実行する処理部を有する。なお、これらの処理は、ハードウェア、またはソフトウェア、またはその組み合わせ構成によって実現される。

【０２３９】

なお、ここでは、暗号処理手段を個別モジュールとした例を示したが、このような独立した暗号処理モジュールを設けず、例えば暗号処理プログラムをＲＯＭに格納し、ＣＰＵ７０１がＲＯＭ格納プログラムを読み出して実行するように構成してもよい。

【０２４０】

乱数発生器７０４は、暗号処理に必要となる鍵の生成などにおいて必要となる乱数の発生処理を実行する。

【０２４１】

送受信部７０５は、外部とのデータ通信を実行するデータ通信処理部であり、例えばリーダライタ等、ＩＣモジュールとのデータ通信を実行し、ＩＣモジュール内で生成した暗号文の出力、あるいは外部のリーダライタ等の機器からのデータ入力などを実行する。

【０２４２】

以上、特定の実施例を参照しながら、本発明について詳解してきた。しかしながら、本発明の要旨を逸脱しない範囲で当業者が実施例の修正や代用を成し得ることは自明である。すなわち、例示という形態で本発明を開示してきたのであり、限定的に解釈されるべきではない。本発明の要旨を判断するためには、特許請求の範囲の欄を参酌すべきである。

【０２４３】

また、明細書中において説明した一連の処理はハードウェア、またはソフトウェア、あるいは両者の複合構成によって実行することが可能である。ソフトウェアによる処理を実行する場合は、処理シーケンスを記録したプログラムを、専用のハードウェアに組み込まれたコンピュータ内のメモリにインストールして実行させるか、あるいは、各種処理が実行可能な汎用コンピュータにプログラムをインストールして実行させることが可能である。例えば、プログラムは記録媒体に予め記録しておくことができる。記録媒体からコンピ

ュータにインストールする他、ＬＡＮ（Ｌｏｃａｌ　Ａｒｅａ　Ｎｅｔｗｏｒｋ）、インターネットといったネットワークを介してプログラムを受信し、内蔵するハードディスク等の記録媒体にインストールすることができる。

【０２４４】

なお、明細書に記載された各種の処理は、記載に従って時系列に実行されるのみならず、処理を実行する装置の処理能力あるいは必要に応じて並列的にあるいは個別に実行されてもよい。また、本明細書においてシステムとは、複数の装置の論理的集合構成であり、各構成の装置が同一筐体内にあるものには限らない。

【産業上の利用可能性】

【０２４５】

上述したように、本発明の一実施例の構成によれば、例えば１バイト単位のデータブロックを配列した長方形配列データを２分割して設定した分割データに対する様々な処理を実行してデータ変換を行う構成において、分割データの一方に対する線形変換処理と、２つの分割データ相互の排他的論理和演算処理と、分割データの一方のデータに対するシフト処理と、２つの分割データのスワップ処理を実行することで、演算コストを低減した効率的なデータ攪拌を実現することができる。さらに、分割データに対する非線形変換や鍵適用演算を含めることでセキュリティレベルの高い暗号処理が実現される。

【図面の簡単な説明】

【０２４６】

【図１】入力データをバイト単位に分割し、バイト単位データを正方形や長方形の配列として配置し、行単位の処理や列単位の処理を繰り返すデータ攪拌に適用する構成について説明する図である。

【図２】ＡＥＳブロック暗号アルゴリズムにおける正方ステートに対する演算例について説明する図である。

【図３】ＡＥＳブロック暗号アルゴリズムにおけるラウンド演算について説明する図である。

【図４】正方ステートに対するラウンド演算によるデータ拡散例について説明する図である。

【図５】長方形配列データ（長方形ステート）に対する演算例について説明する図である。

【図６】長方形配列データ（長方形ステート）に対するラウンド演算例について説明する図である。

【図７】長方形ステートに対するラウンド演算によるデータ拡散例について説明する図である。

【図８】長方形ステートの半分の分割データに対する処理例について説明する図である。

【図９】長方形ステートに対するラウンド演算処理例について説明する図である。

【図１０】長方形ステートに対するラウンド演算処理によるデータ拡散例について説明する図である。

【図１１】長方形ステートに対するラウンド演算処理によるデータ拡散例について説明する図である。

【図１２】本発明の第１実施例におけるラウンド演算の構成例について説明する図である。

【図１３】本発明の第１実施例におけるラウンド演算において実行するデータ処理の例について説明する図である。

【図１４】本発明の第１実施例のラウンド演算処理によるデータ拡散例について説明する図である。

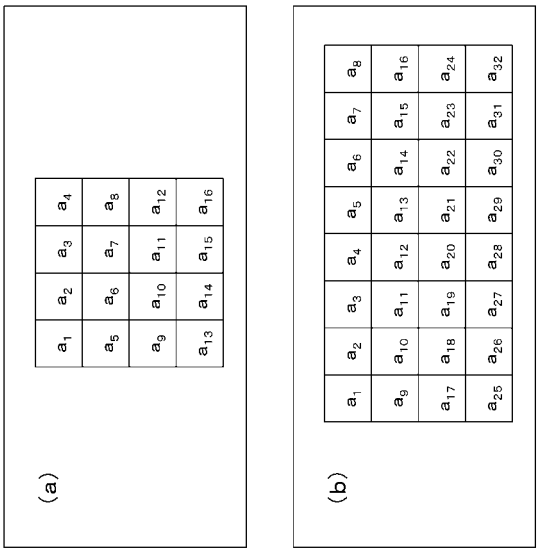
【図１５】本発明の第１実施例のラウンド演算処理によるデータ拡散例について説明する図である。

【図１６】本発明の第２実施例におけるラウンド演算の構成例について説明する図である。

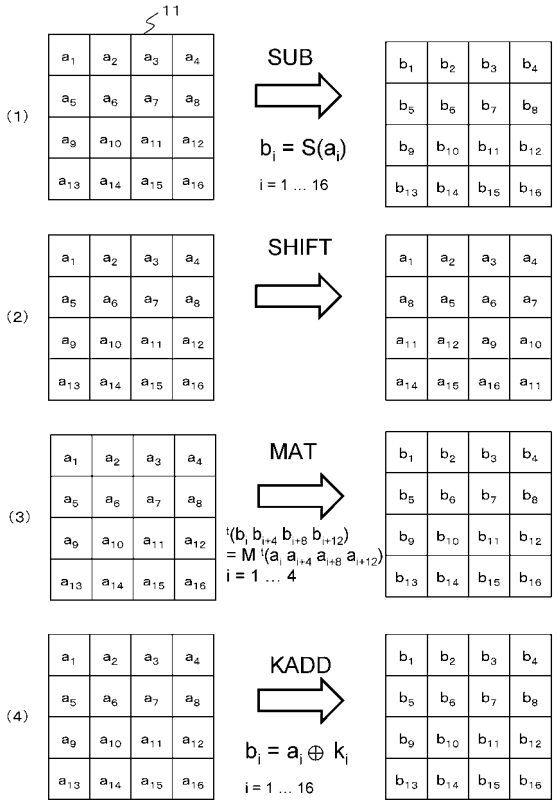
【図１７】本発明の第２実施例におけるラウンド演算において実行するデータ処理の例について説明する図である。
【図１８】本発明の第２実施例のラウンド演算処理によるデータ拡散例について説明する図である。
【図１９】本発明の第２実施例のラウンド演算処理によるデータ拡散例について説明する図である。
【図２０】本発明の第３実施例におけるラウンド演算の構成例について説明する図である。
【図２１】本発明の第３実施例におけるラウンド演算において実行するデータ処理の例について説明する図である。
【図２２】本発明の第３実施例のラウンド演算処理によるデータ拡散例について説明する図である。
【図２３】本発明の第３実施例のラウンド演算処理によるデータ拡散例について説明する図である。
【図２４】本発明の第４実施例におけるラウンド演算の構成例について説明する図である。
【図２５】本発明の第４実施例におけるラウンド演算において実行するデータ処理の例について説明する図である。
【図２６】本発明の第４実施例のラウンド演算処理によるデータ拡散例について説明する図である。
【図２７】実施例１におけるシフト処理（Ｈ－ＳＨＩＦＴ）を長方形ステートの左右半分の分割データの両者に適用する処理のアルゴリズムについて説明する図である。
【図２８】全体で２ｍｎ個のバイト単位データを持つデータを、２ｎ列、ｍ行のa_1～a_{2mn}の２ｍｎ個のバイト単位データを有する長方形ステートとした一般形に対するデータ変換処理について説明する図である。
【図２９】全体で２ｍｎ個のバイト単位データを持つデータを、２ｎ列、ｍ行のa_1～a_{2mn}の２ｍｎ個のバイト単位データを有する長方形ステートとした一般形に対するデータ変換処理について説明する図である。
【図３０】長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステートに対する実施例１によるデータ拡散例を説明する図である。
【図３１】長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステートに対する実施例２によるデータ拡散例を説明する図である。
【図３２】長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステートに対する実施例３によるデータ拡散例を説明する図である。
【図３３】長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステートに対する実施例４によるデータ拡散例を説明する図である。
【図３４】長方形ステートの半分の分割データが正方ステートとならない一般化された長方形ステートに対する実施例４によるデータ拡散例を説明する図である。
【図３５】本発明に係る処理を実行するデータ変換装置としてのＩＣモジュールの構成例を示す図である。
【符号の説明】
【０２４７】
１１　正方形配列データ（正方ステート）
２１　正方ステート
３１　バイト単位データ
５１　長方形配列データ（長方形ステート）
６１　長方形配列データ（長方形ステート）
７１　バイト単位データ
１００　長方形ステート
１０１　バイト単位データ

１０２　バイト単位データ
２００　長方形ステート
２０１　バイト単位データ
２０２　バイト単位データ
２２０　長方形ステート
２２１　バイト単位データ
２２２　バイト単位データ
２３０　長方形ステート
２３１　バイト単位データ
２３２　バイト単位データ
２４０　長方形ステート
２４１　バイト単位データ
２４２　バイト単位データ
３１０　長方形ステート
３１１　バイト単位データ
３２０　長方形ステート
３２１　バイト単位データ
３３０　長方形ステート
３３１　バイト単位データ
３４０　長方形ステート
３４１　バイト単位データ
７００　ＩＣモジュール
７０１　ＣＰＵ(Central processing Unit)
７０２　メモリ
７０３　データ変換部
７０４　乱数発生器
７０５　送受信部

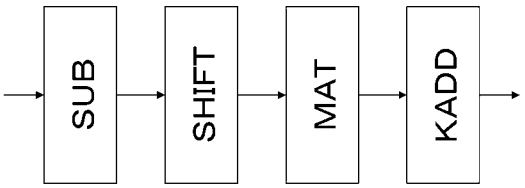
【図１】


(a)
(b)

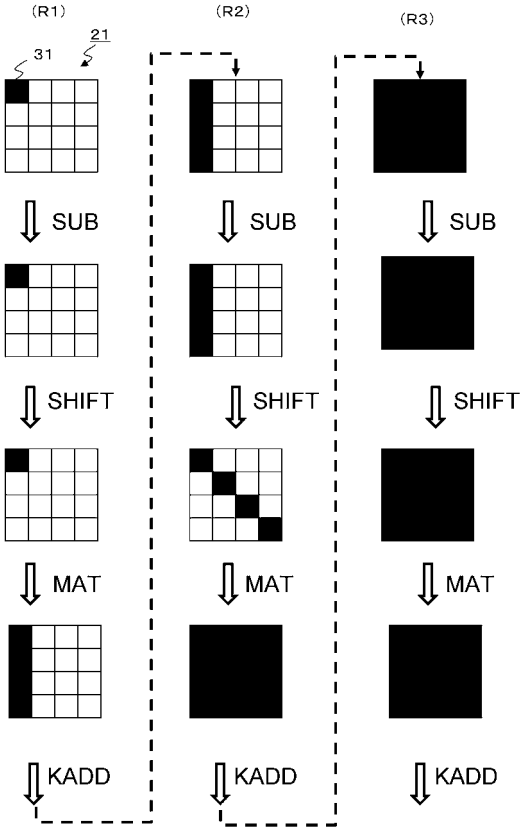
【図２】


11
(1)
SUB
$b_i = S(a_i)$
i = 1 … 16
(2)
SHIFT
(3)
MAT
$^t(b_i\ b_{i+4}\ b_{i+8}\ b_{i+12}) = M\ ^t(a_i\ a_{i+4}\ a_{i+8}\ a_{i+12})$
i = 1 … 4
(4)
KADD
$b_i = a_i \oplus k_i$
i = 1 … 16

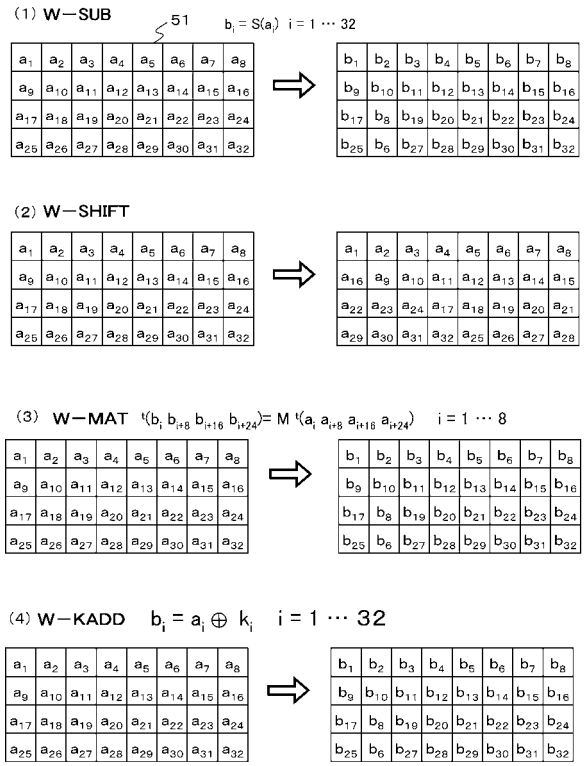
【図３】


SUB
SHIFT
MAT
KADD

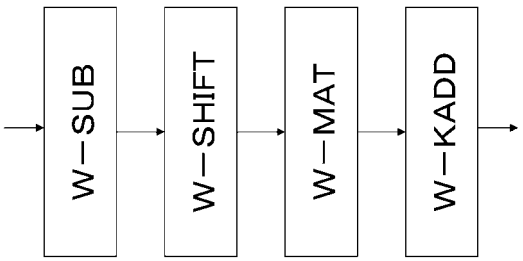
【図４】


(R1)
(R2)
(R3)
21
31
SUB
SHIFT
MAT
KADD

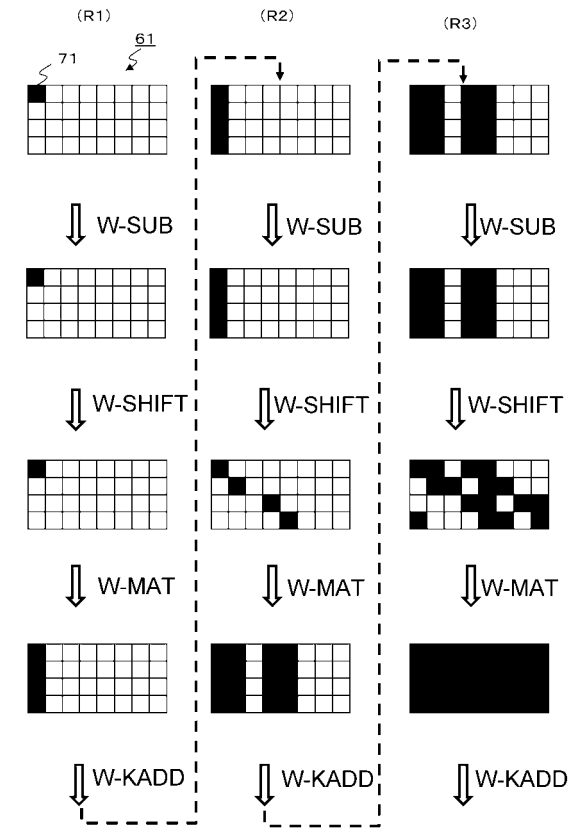
【図５】


(1) W−SUB
51
$b_i = S(a_i)$ i = 1 … 32
(2) W−SHIFT
(3) W−MAT $^t(b_i\ b_{i+8}\ b_{i+16}\ b_{i+24}) = M\ ^t(a_i\ a_{i+8}\ a_{i+16}\ a_{i+24})$ i = 1 … 8
(4) W−KADD $b_i = a_i \oplus k_i$ i = 1 … 32

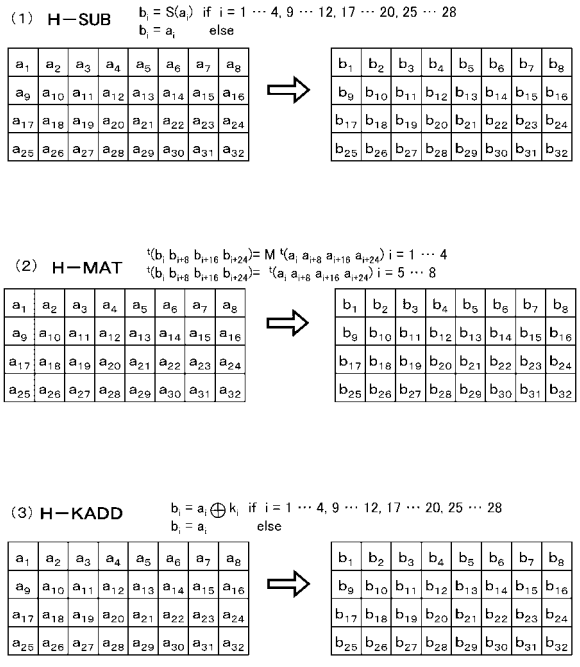
【図６】


W−SUB
W−SHIFT
W−MAT
W−KADD

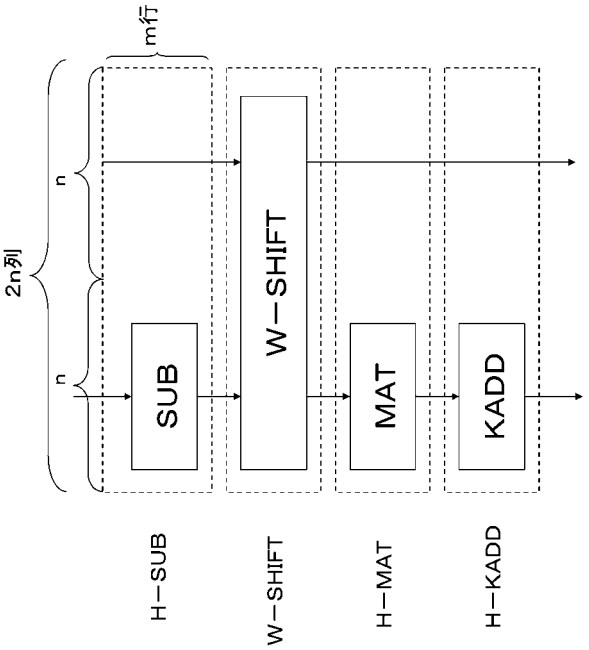
【図７】


(R1)
(R2)
(R3)
61
71
W-SUB
W-SHIFT
W-MAT
W-KADD

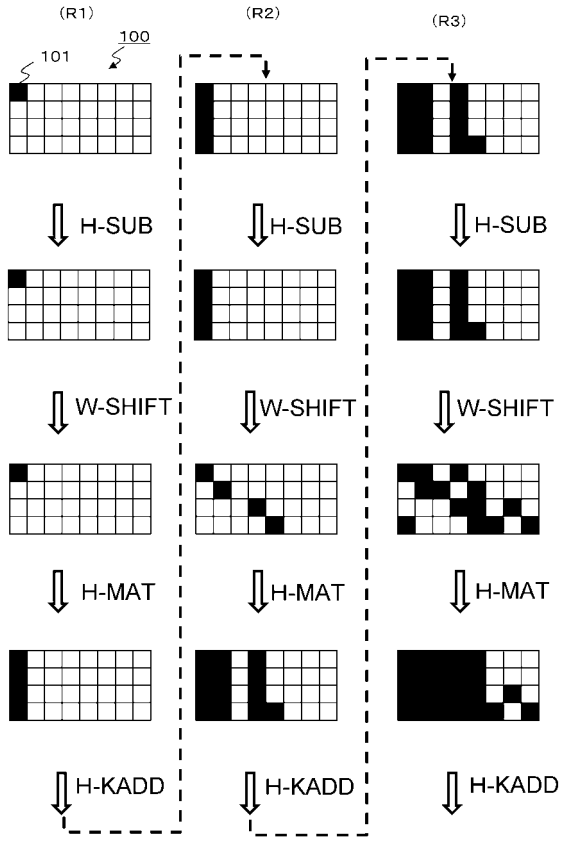
【図８】


(1) H−SUB
$b_i = S(a_i)$ if i = 1 … 4, 9 … 12, 17 … 20, 25 … 28
$b_i = a_i$ else
(2) H−MAT
$^t(b_i\ b_{i+8}\ b_{i+16}\ b_{i+24}) = M\ ^t(a_i\ a_{i+8}\ a_{i+16}\ a_{i+24})$ i = 1 … 4
$^t(b_i\ b_{i+8}\ b_{i+16}\ b_{i+24}) = {}^t(a_i\ a_{i+8}\ a_{i+16}\ a_{i+24})$ i = 5 … 8
(3) H−KADD
$b_i = a_i \oplus k_i$ if i = 1 … 4, 9 … 12, 17 … 20, 25 … 28
$b_i = a_i$ else

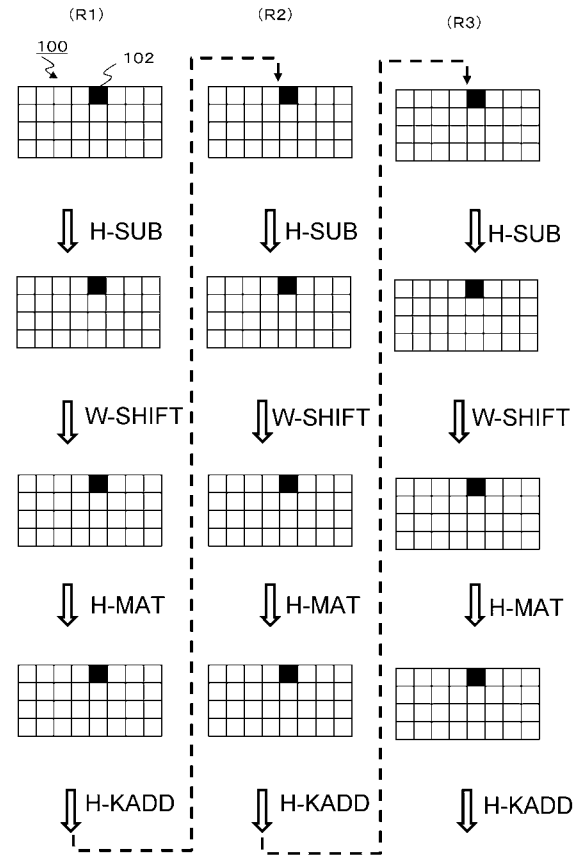
【図９】



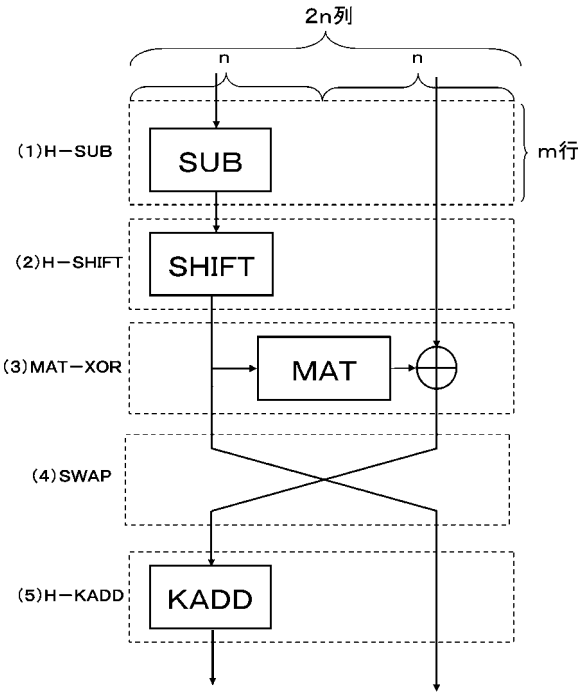
【図１０】



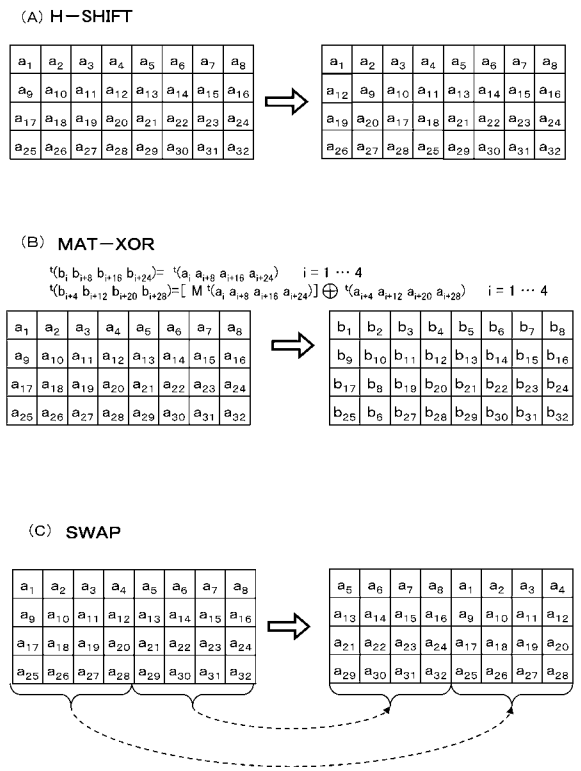
【図１１】



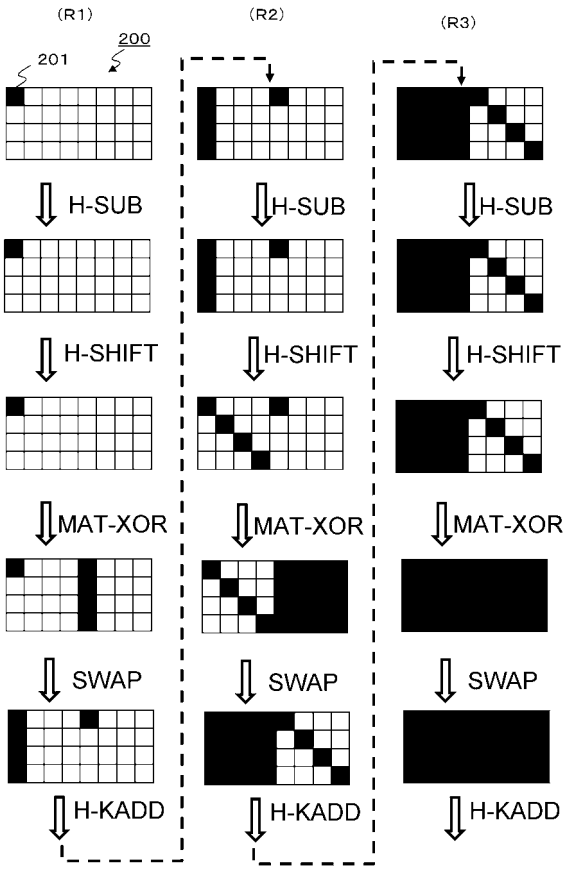
【図１２】



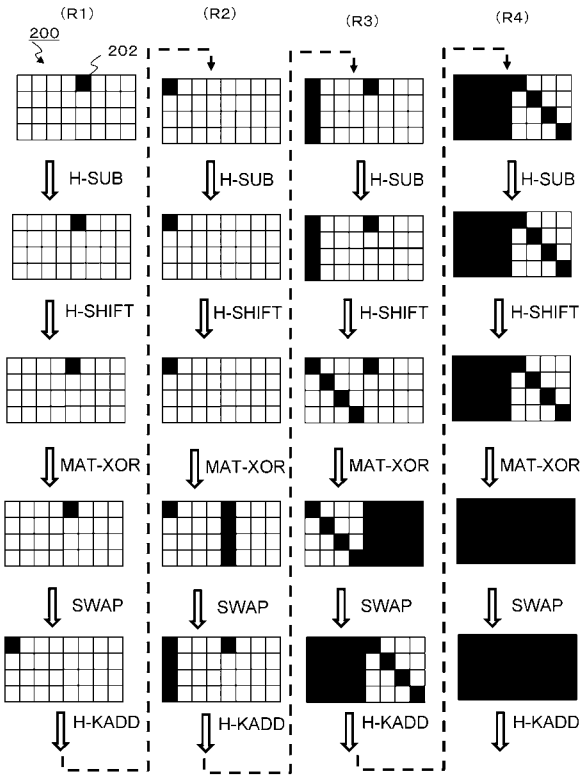
【図１３】



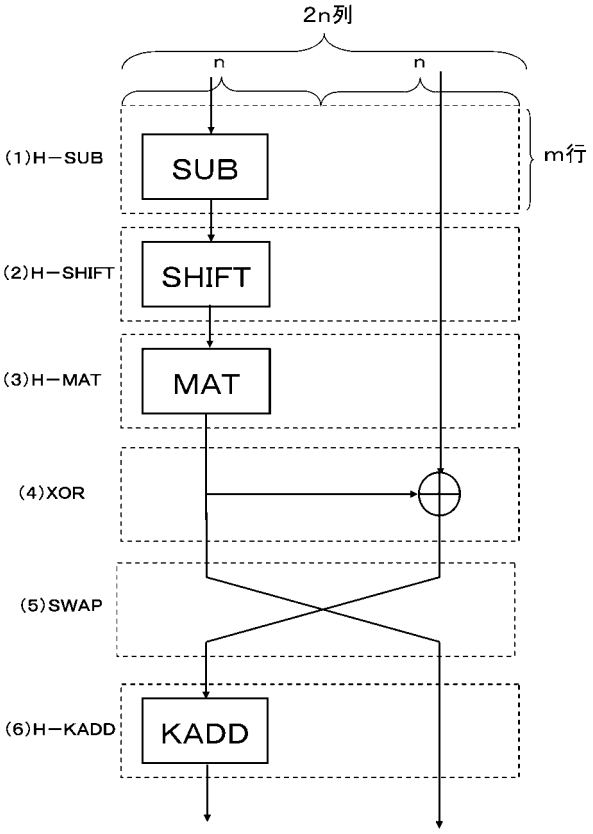
【図１４】



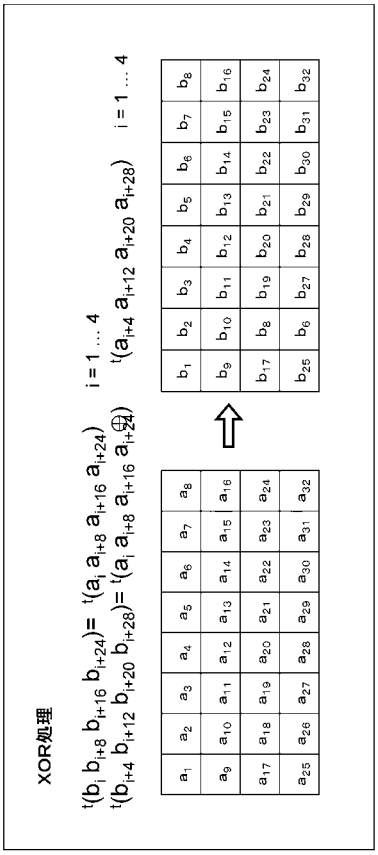
【図１５】



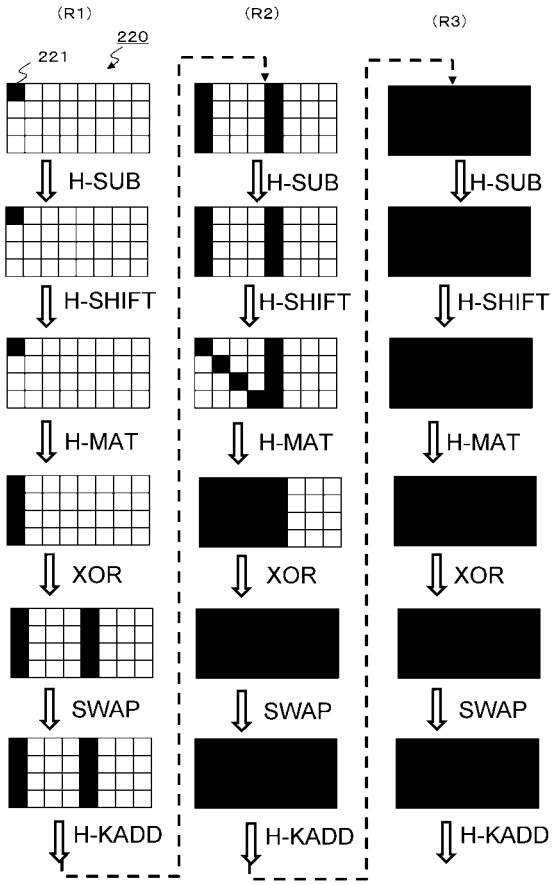
【図１６】



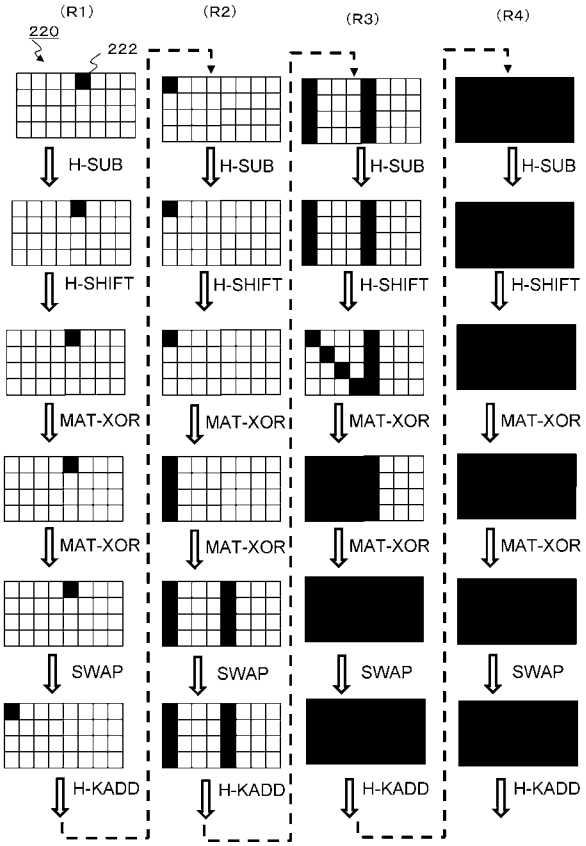
【図１７】



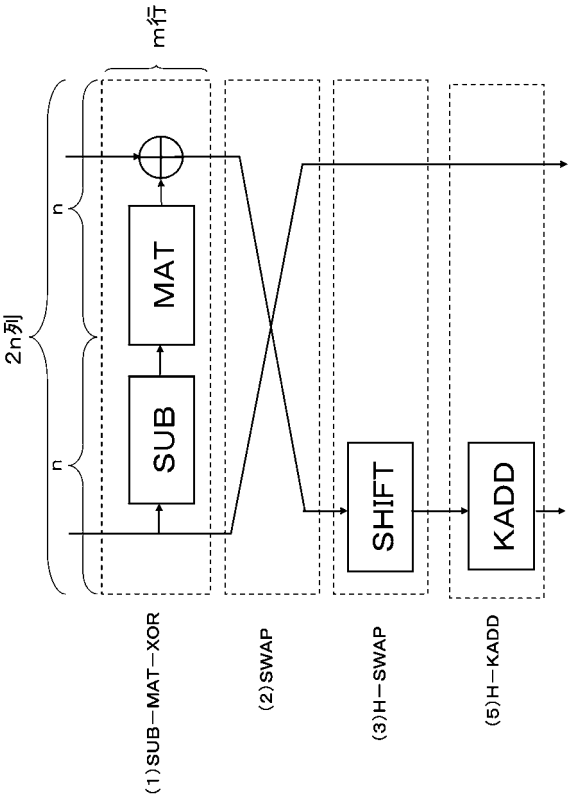
【図１８】



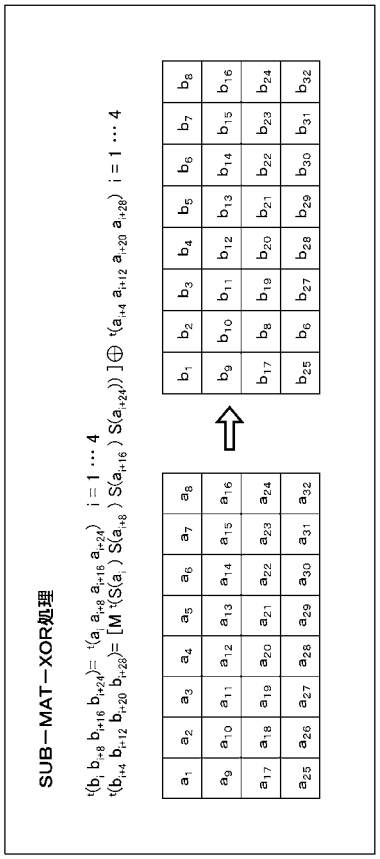
【図１９】



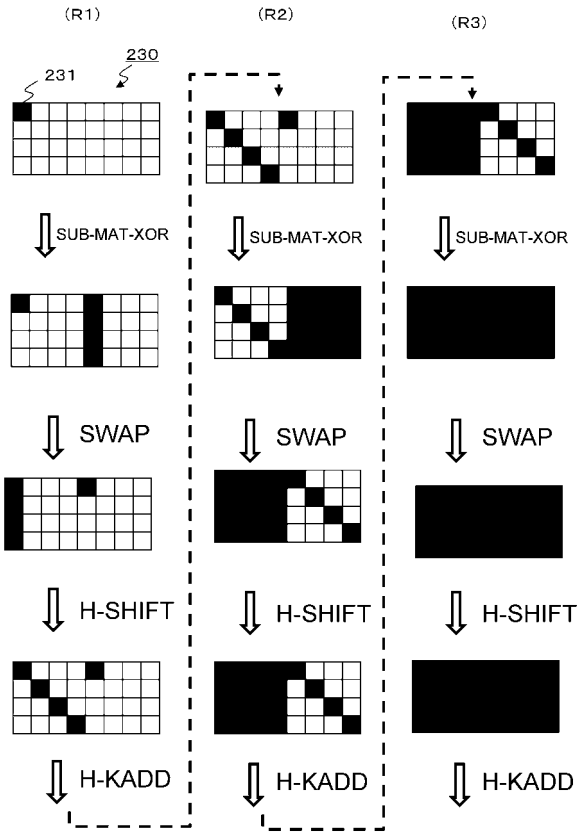
【図２０】



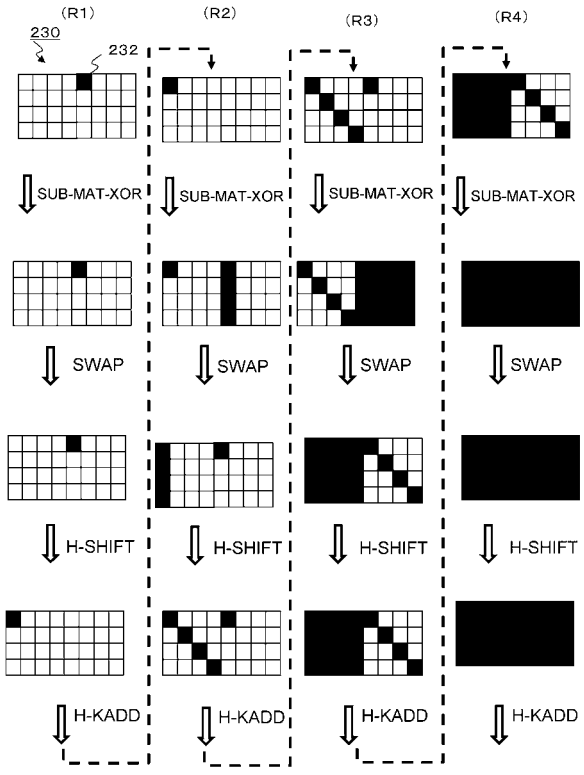
【図２１】



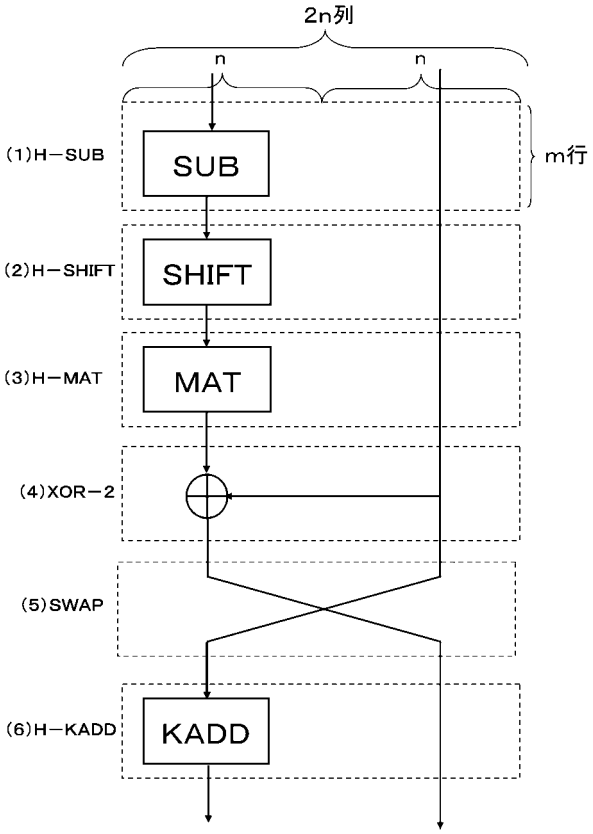
【図２２】



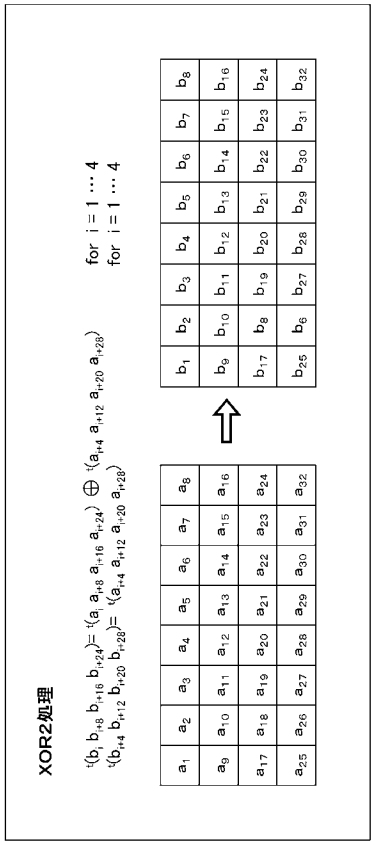
【図２３】



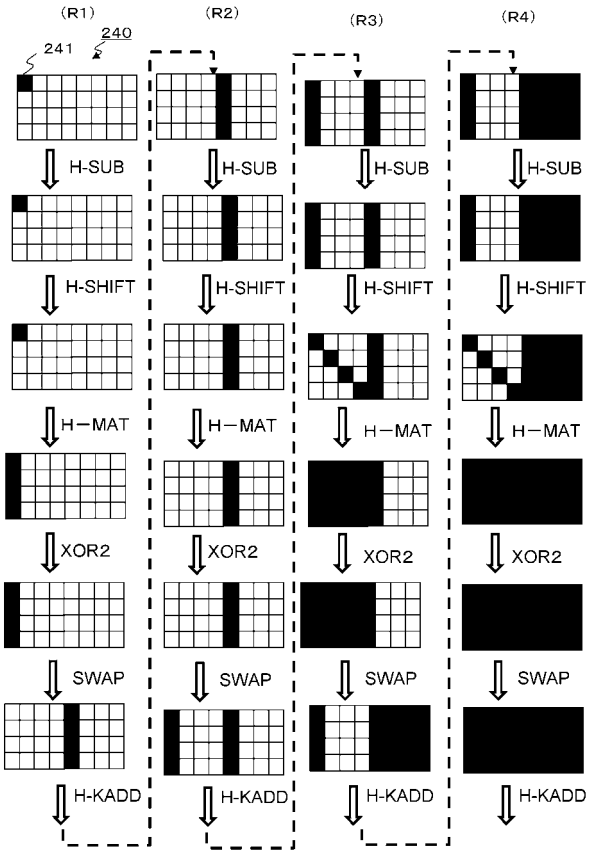
【図２４】



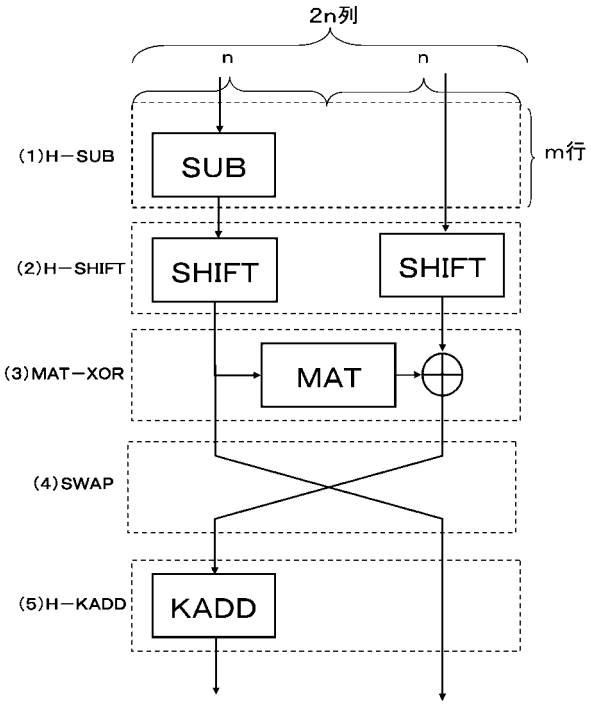
【図２５】



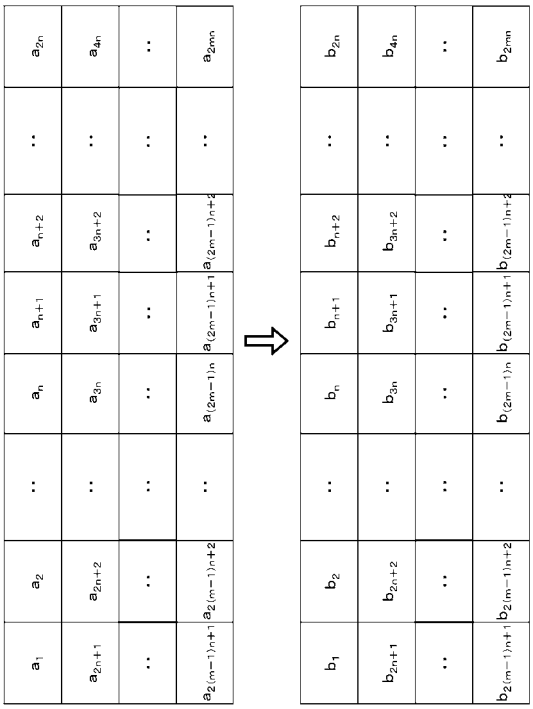
【図２６】



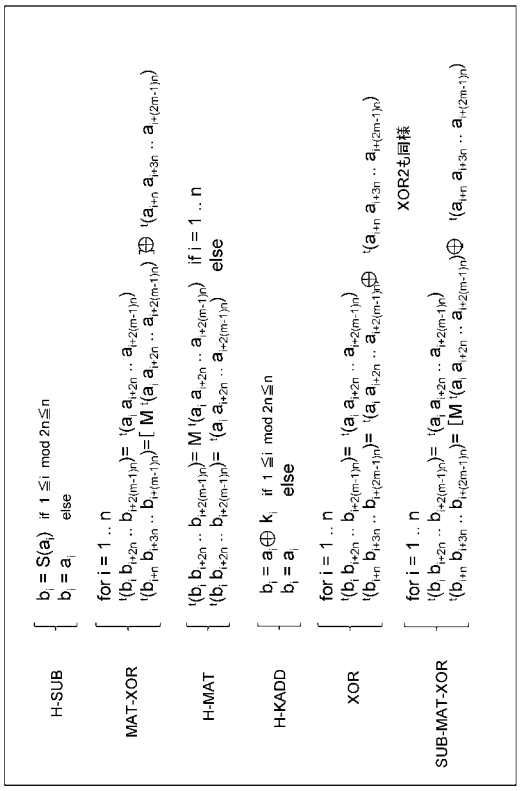
【図２７】



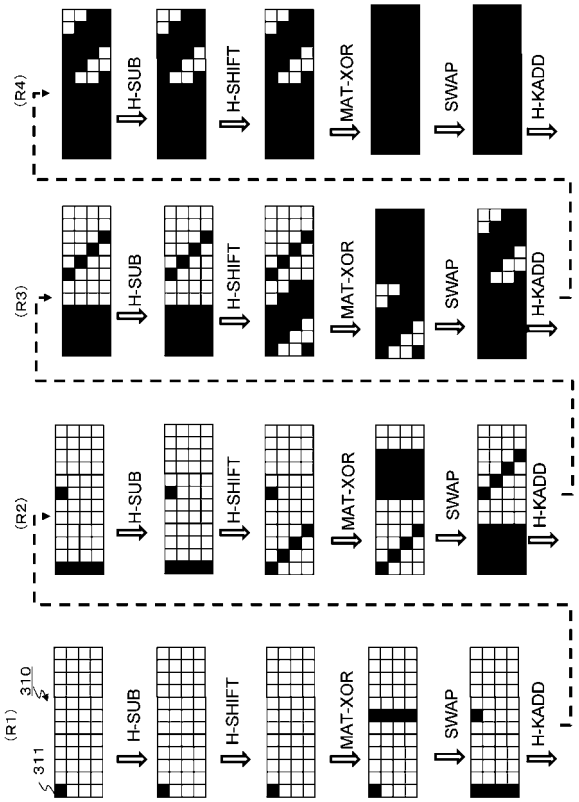
【図２８】



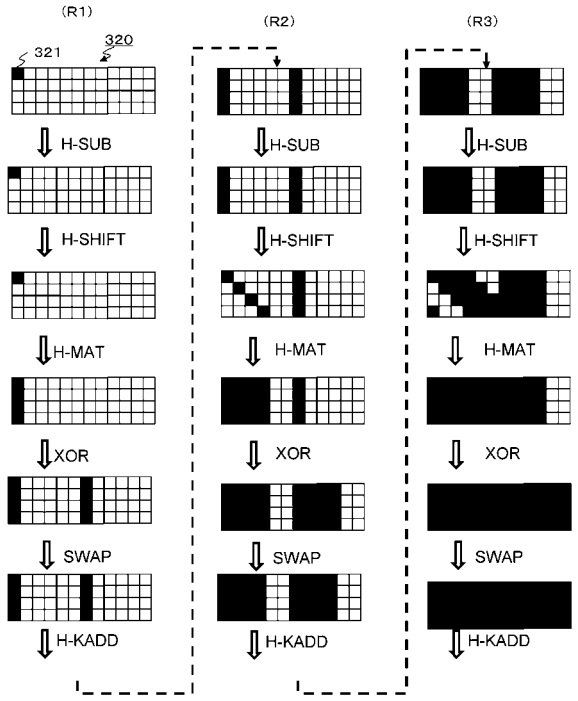
【図２９】



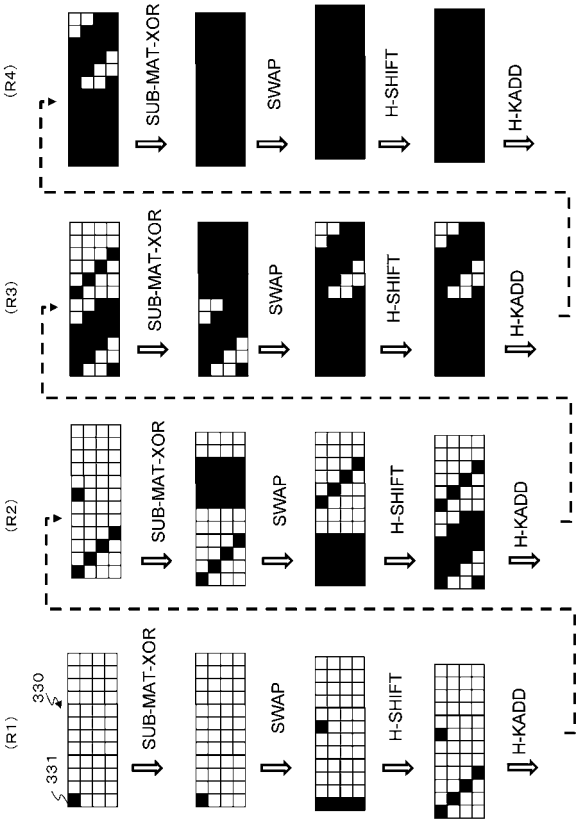
【図３０】



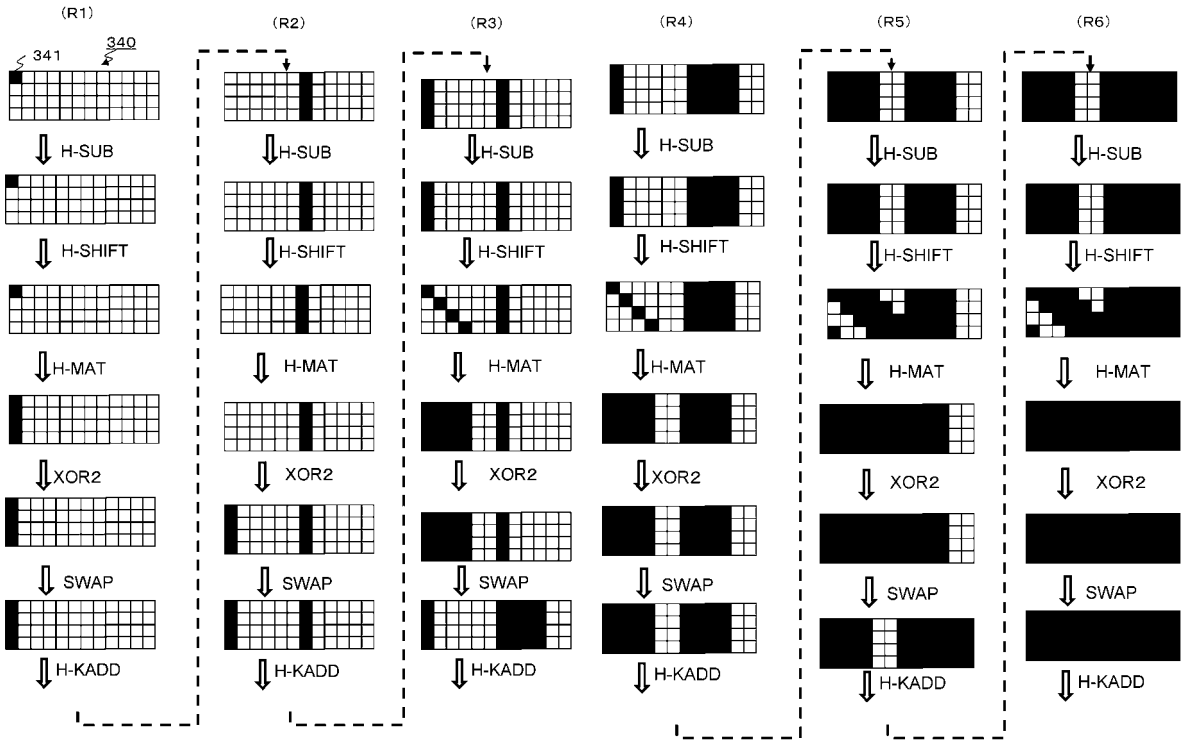
【図３１】



【図３２】

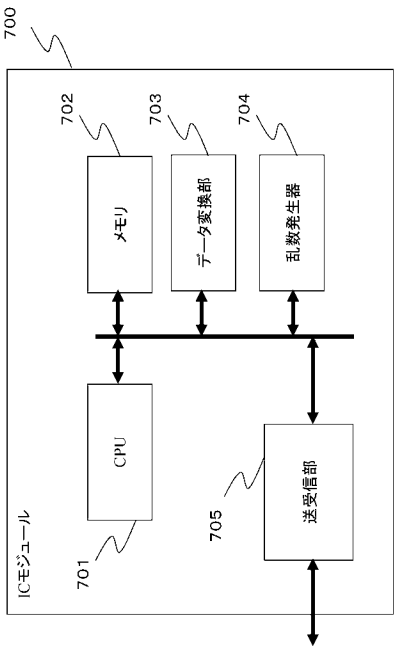


【図３３】


(R1)
341
340
H-SUB
H-SHIFT
H-MAT
XOR2
SWAP
H-KADD
(R2)
(R3)
(R4)
(R5)
(R6)

【図３４】

【図３５】


700
702
703
704
メモリ
データ変換部
乱数発生器
CPU
705
送受信部
701
ICモジュール

フロントページの続き

(72)発明者　渋谷　香士
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　盛合　志帆
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　秋下　徹
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　岩田　哲
愛知県名古屋市千種区不老町１番　国立大学法人名古屋大学内

審査官　金沢　史明

(56)参考文献　J. Daemen and V. Rijmen, AES Proposal: Rijndael, １９９９年　9月　3日, pp. 1-16,25-29,42
白井太三, 他, １２８ビットブロック暗号ＣＬＥＦＩＡ, 電子情報通信学会技術研究報告, 社団法人電子情報通信学会, ２００７年　5月１１日, vol. 107, no. 44, pp. 1-9
岩田哲, 他, ハッシュ関数ファミリーＡＵＲＯＲＡ, 電子情報通信学会技術研究報告, 社団法人電子情報通信学会, ２００９年　3月　2日, vol. 108, no. 473, pp. 277-286

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０９Ｃ　　１／００
Ｈ０４Ｌ　　９／０６