ハッシュ関数：算術演算型とテーブル参照型の比較検討

松井　充

三菱電機情報技術総合研究所

247 鎌倉市大船 5-1-1
Tel: 0467-41-2181 Fax: 0467-41-2138
matsui@iss.isl.melco.co.jp

あらまし：現在ハッシュ関数の構成法として、ブロック暗号を組み合わせる方式と、MD5 に代表される MD 方式の２種類が広く用いられている。本稿ではこれらの特性を比較検討するとともに、並列構造をもった新しいハッシュ関数の構成原理について、現在までの検討結果を報告する。この方式は特にハードウエアで高速な処理を実現し、例えば 0.5μ CMOS gate-array では 1.5Gbps から 2.0Gbps の速度を達成することが期待できる。

Hash Functions : Comparisons between Arithmetic Operations and Loop-up Tables

Mitsuru MATSUI

Information Technology R&D Center
Mitsubishi Electric Corporation

5-1-1 Ofuna Kamakura Kanagawa 247 Japan
TEL: +81-467-41-2181 FAX: +81-467-41-2138
matsui@iss.isl.melco.co.jp

Abstract: In this article we discuss advantages and disadvantages of two methods for designing hash functions: a combination of block ciphers and an MD5-based construction. We also propose a prototype of a new hash function from a viewpoint of fast processing using its parallel architecture. This function is particularly fast on hardware; for example, its throughput is expected to be 1.5Gbps to 2.0Gbps by a process of 0.5μ CMOS gate-array.

1　ハッシュ関数

ハッシュ関数 (Hash Function) とは、任意長のメッセージを固定長のデータに圧縮する非可逆な関数であり、暗号技術においては、ハッシュ関数はパスワード認証・ディジタル署名・メッセージ認証などの目的で幅広く用いられている。例えばパスワード認証においては、計算機は各ユーザーのパスワードをあらかじめ何らかの形式で記憶しておく必要があるが、これをそのまま格納するのは安全上望ましくないため、ハッシュ関数で一旦変換してから保存しておくのが普通である。

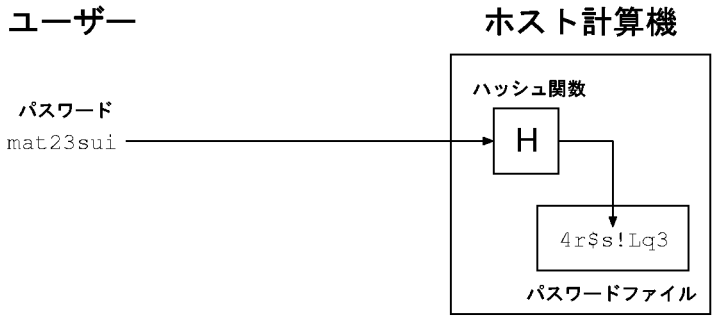

図１：パスワードはハッシュ関数で変換されたのち計算機に保存される

ハッシュ関数 $H(M)$ の安全性のうち最も重要なものは非衝突一致性 (Collision Resistance) である。非衝突一致性とは、次の関係式を満たす異なるメッセージ M, M' を見つけることが困難なことと定義される。この性質は、言い替えれば正しいメッセージ M からにせのメッセージ M' を作り出すことが難しいということを意味している。

$$M \neq M' \quad \text{かつ} \quad H(M) = H(M') \tag{1}$$

ハッシュ関数はそもそも圧縮関数であるため、実際にはには上の関係式を満たすような異なるメッセージ M と M' の組は多数存在する。しかしこのような組を見つけ出すことが計算量的に困難である時、すなわち現実的に実現不可能なほど莫大な計算が必要である時、そのハッシュ関数は非衝突一致性をもつというのである。

ところでハッシュ値（ハッシュ関数の出力値）の長さはその安全性の重要なパラメータの一つである。一般に、ハッシュ値の長さが n ビットであるとき、およそ $2^{n/2}$回程度ハッシュ関数を計算すれば、上の関係式を満たすメッセージ M と M' の組を見いだせることが知られている。この攻撃法は birthday attack と呼ばれるもので、したがってハッシュ値はある程度長くなければならない。現在実用化されているハッシュ関数は、128 ビットから 196 ビット程度が普通である。

ハッシュ関数の構成法として現在２種類がよく知られている。そのひとつはブロック暗号を組み合わせる方法であり、もうひとつは MD5 に代表される MD 方式である。そこで次節と次々節ではこれらの２つの方式を紹介するとともに、その長所と短所について考察を加える。

2　ブロック暗号に基づくハッシュ関数

暗号化鍵 K および（固定長）メッセージ M をもつブロック暗号を $B_K(M)$ とし、関数 H を $H(M) = B_K(M) \oplus M$ と定義する時（今 K は任意に固定されているものとする）、ブロック暗号 B が安全であるならば H は非衝突一致性をもつと考えられている。この原理に基づいて安全なハッシュ関数を構成しようというのがブロック暗号に基づくハッシュ関数構成の基本的な考え方である。ただしこの例ではメッセージ M の長さはブロック暗号のブロック長に限定されてしまうので、ハッシュ関数としてこれを例えば以下のように可変長メッセージを処理できるよう拡張するのである。

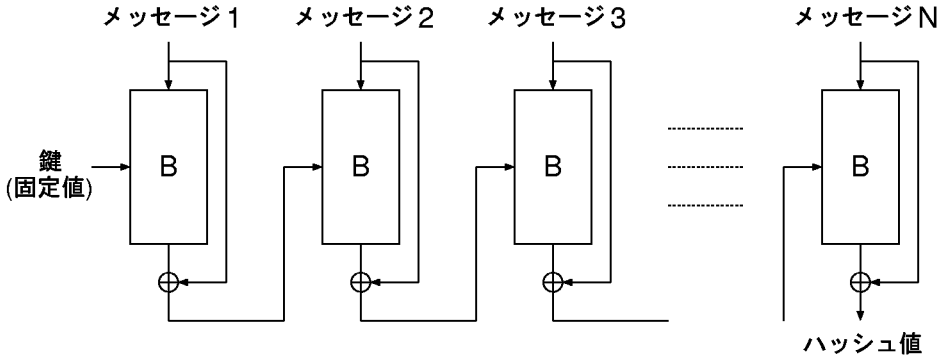

図２：ブロック暗号に基づくハッシュ関数の基本形

この方式は以下のような長所と短所を持っている

長所 1： ハッシュ関数の安全性をブロック暗号の安全性の問題に帰着させることができる。つまりこれまでのブロック暗号の安全性に関する豊富な結果を利用することができる。

長所 2： ブロック暗号の並列構造を利用して速度を高めることができる。これはブロック暗号を組み合わせてハッシュ関数を構成する時に特に有効である。

短所 1： ハッシュ値のサイズをブロック暗号のブロックサイズと同じにすると強度上不十分な場合が多いため、２つ以上のブロック暗号を組み合わせることが必須である。

短所 2： 一般に MD 方式に基づくハッシュ関数に比べて総計算量が多く、並列処理が不可能な環境では高速性を達成するのが困難なことが多い。

ハッシュ値の長さについては、例えばブロック暗号として現在最も一般的である 64 ビットブロック暗号をそのままハッシュ関数の基本ブロック B に適用した場合、前節の birthday attack を用いれば 2^{32} 回程度のブロック暗号の計算で衝突するメッセージ M と M' を見つけることができる。しかしながら 2^{32} 回という計算量は、現在ではパーソナルコンピューターでも数時間で実現可能であるため、関数 B はいくつかのブロック暗号を組み合わせてハッシュ値の長さを 128 ビット以上に設計するのが一般的である。これらの方式のいくつかはすでに ISO でも規格化されている [1]。なお最新の研究結果については文献 [2] で知ることができる。

3　MD 方式に基づくハッシュ関数

これに対して MD4 [3] にはじまる新しいハッシュ関数の構成法が最近ではよく使われている。この基本的な考え方は、ブロック暗号の暗号化鍵に相当する部分（正確には拡大鍵に相当する部分）にメッセージを入力するというものである。通常、ブロック暗号の拡大鍵のサイズはブロックサイズに比べて相当大きいが、このことはハッシュ関数のひとつの基本ブロックあたりに扱えるメッセージのサイズが大きいことを意味している。したがってハッシュ関数の処理が高速化されるのである。その基本形は以下のようなものである。

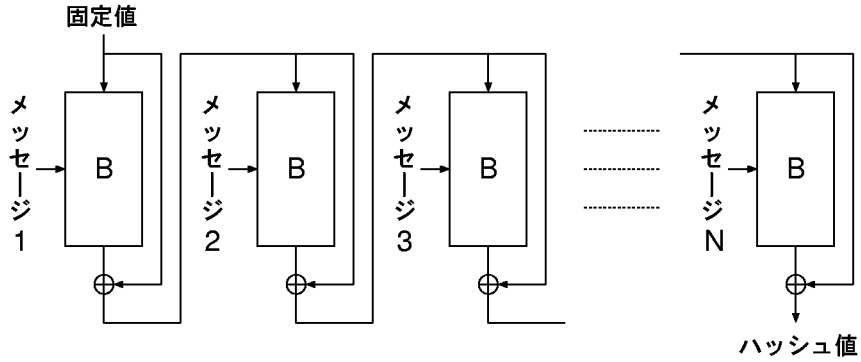

図 3 ：MD 方式に基づくハッシュ関数の基本形

この方式は以下のような長所と短所を持っている

長所 1： 構成原理を大きく変化させることなく、比較的容易にハッシュ値のサイズをコントロールすることができる。

長所 2： 一般にソフトウエア（特に一度に扱えるビット幅の長い 32 ビットあるいは 64 ビットプロセッサ）では高速な処理を実現することができる。

短所 1： ブロック暗号に基づく方式に比べて安全性の理論的根拠がまだ未確立であり、安全性評価より方式提案が先行している状況にある。

短所 2： ひとつの基本ブロックあたり長いメッセージを扱うため、安全性を考慮すると総計算量を増やすことなく並列性を高めることは難しい。

MD 方式に基づくハッシュ関数は、32 ビットあるいは 64 ビットプロセッサ上のソフトウエアで最高の性能が実現されることを前提として設計されることが多い。これらのタイプのハッシュ関数で良く知られているものとして次がある。（なお MD4 は Hans Dobbertin によって衝突するメッセージが発見されている [7]）。

MD5 [4]	ハッシュ値 128 ビット	RFC1321
SHA [5]	ハッシュ値 160 ビット	FIPS180-1
RIPEMD-160 [6]	ハッシュ値 160 ビット	

4　新しいハッシュ関数の設計指針

MD 方式に基づくハッシュ関数は、通常演算の単位が 32 ビットあるいは 64 ビットのプロセッサ上のソフトウエアで最高の性能が出るように設計されているので、このプラットフォーム上ではブロック暗号に基づくハッシュ関数に比べてきわめて高速に計算を実行することができるが、それ以外のたとえば８ビットのマイクロプロセッサでは 32 ビットシフト演算に時間がかかるなどのペナルティがある。また MD 方式に基づくハッシュ関数は並列性をあまり高くできないため、たとえばハードウエアで構成した場合では大きな高速化が望めないと考えられる。

そこで今回は、高い並列処理構造を導入することにより、特定のプロセッサ上のソフトウエアで最高の性能を発揮することよりも、あらゆるプラットフォームに適した、特にハードウエアでも高いパフォーマンスを実現できるハッシュ関数の構成法を考えた。ところが前節で見たように、並列処理構造を最大限に生かすためにはブロック暗号に基づく構成をとるべきであるが、これは一般に処理スピードに限界があり、またブロック暗号のロジックを一律に削減してしまうと安全上の問題を生じることとなる。

この問題を解決するため、最終ブロックの処理が終ったのちに、再びこれまでのすべてのメッセージブロックに依存する新しい追加の処理を付け加えることを提案する。その方法の基本形は以下のとおりである。

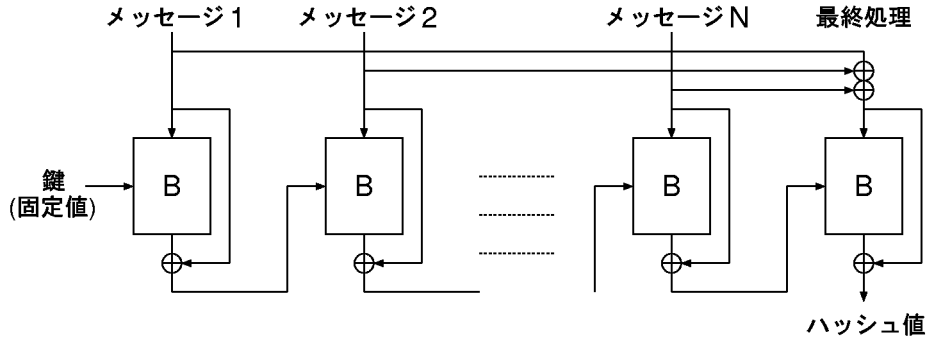

図４：新しいハッシュ関数の基本形

この例では、最終ブロックの処理のあとに、それまでのメッセージをすべて排他的論理和したものを新たなメッセージとしてもう一度基本ブロックの処理を行ない、その結果を最終的なハッシュ値としている。このような処理にした理由は、たとえ中間ブロックで衝突があったとしてもハッシュ値が同じになるためには追加ブロックのところでもう一度衝突がおこならくてはならない。また追加ブロックでの入力を変えずに衝突をおこすためには中間ブロックのうち少なくとも２箇所で入力が変化する必要がある。このような条件でハッシュ値の衝突を見つけ出すのは難しいと考えられるからである。この困難さの代償としてひとつの基本ブロックのサイズを小さくとることができると期待される。

また速度の点では、追加ブロックは冗長であるが、この冗長度はメッセージの長さに関わらず一回だけであるので、その速度低下は漸近的に無視できる。

5　基本ブロック B の設計例

本節では、160 ビットのハッシュ値をもつハッシュ関数の基本ブロックの例を与える。以下に示す関数 B は、160 ビットの入出力と 160 ビットのパラメータ（これはブロック暗号における暗号化鍵あるいは拡大鍵に対応する）をもつものである。

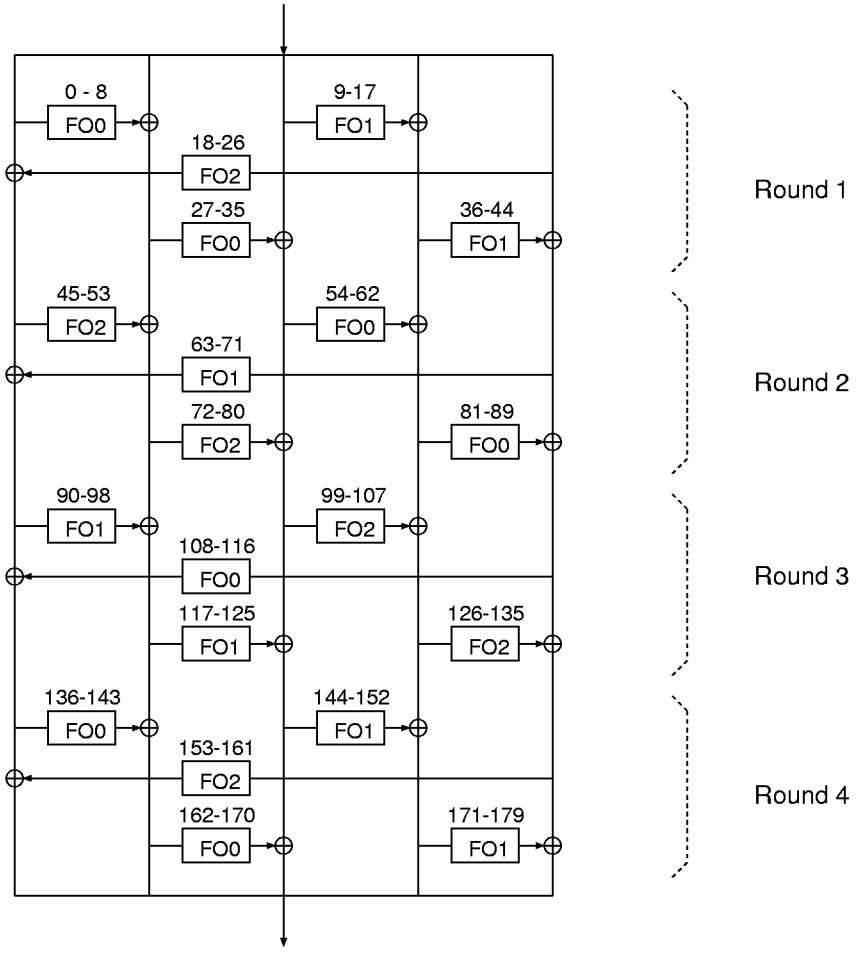

図５：基本ブロック B の設計例

ここで160 ビットの入力は32 ビットごとに5 分割され、３種類の FO 関数 (FO_1, FO_2, FO_3) を合計 20 回通して変換される。この結果は最後に結合されて再び 160 ビットとして出力される。各 FO 関数は、160 ビット（20 バイト）のパラメータのうち 9 バイトを用いて 32 ビットの入力を別の 32 ビットに変換する。図において各関数の上部に示した数字は、パラメータの何バイト目から何バイト目までを用いるかを表している。ここで最初のバイトは第０バイトとし、さらにこれらの値は 20 を法として考えるものとする。例えば 18–26 とは、18,19,0,1,2,3,4,5,6 の各バイトを意味している。

この構成では３つの FO関数を完全に並列に計算することができる。したがって並列構造の高いプロセッサや、あるいはハードウエア上ではきわめて高速な演算が可能になることが期待できる。

6　関数 FOの構成例

ここでは 32 ビットの入出力と 9 バイトのパラメータをもつ関数 FOの例を与える。以下では、差分解読法と線形解読法に対する証明可能安全性をもったブロック暗号 MISTY の構成要素を利用している。

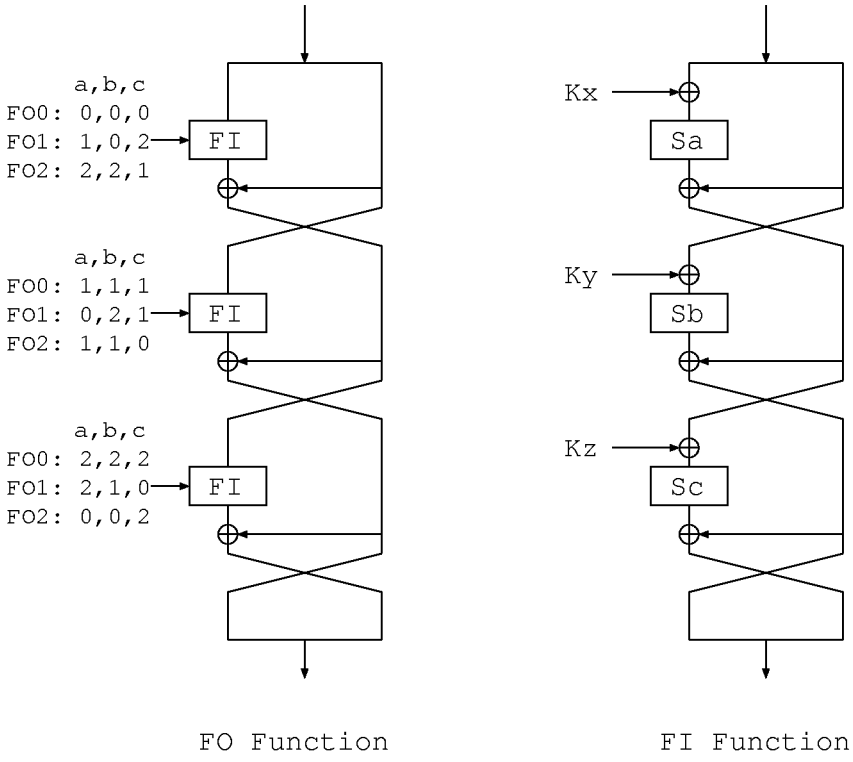

図６：関数 FOの設計例

FO は入力された 32 ビットを 16 ビットづつに分割し、FI を３回作用させたのち、最後に結合して 32 ビットを出力する。また FI は入力された 16 ビットを 8 ビットづつに分割し、置換表 S_0,S_1,S_2 を以下に示す順序で合計３回作用させたのち、最後に結合して 16 ビットを出力する。ここでどの置換表をどの場所で作用させるかは FO の種類、FI の位置、置換表の位置によってそれぞれ異なっている。その規則は図の FO の左側に記述されている。例えば FO_0 の最初の FI に含まれる置換表にはすべて S_0 が利用される。また FO_1 の最後の FI に含まれる置換表は順に S_2,S_1,S_0 が利用される。またパラメータ情報は、先に与えた９バイトを３つの FI 関数の K_x,K_y,K_z に順に作用させる。

また置換表の詳細は未定義であるが、その設計原理としては、MISTY と同じく差分確率や線形確率の低いものを利用し、また S_1 と S_2 はそれぞれ S_0 の出力を右にそれぞれ 1 または 2 ビット回転シフトさせたものをとることでテーブルサイズを削減することができる。なおこの例では、４つの置換テーブルを完全に並列に処理できることに注意しよう。したがって本稿で提案する 160 ビットハッシュ関数全体では１２個の置換テーブルを完全に並列処理できるのである。

7　まとめと性能評価

本稿ではハッシュ関数の２つの代表的な構成法を比較し、さらに新しいハッシュ関数の構成原理について述べた。例に示したアルゴリズムのハードウエアでの速度は置換表の内容にも依存するが、ハードウエア用に考慮されたものであれば 1.5Gbps〜2Gbps (0.5μ CMOS gate-array）程度の性能が出ると期待される。

一方ソフトウエアでは、MD シリーズのターゲットプラットフォームである 32 ビットマイクロプロセッサ上で、速度をさらに向上させるようアルゴリズムの調整をおこなう予定である。

参考文献

[1] ISO/IEC 10118, "Information technology – Security Techniques – Hash-functions, Part 1: General and Part 2: Hash-functions using an n-bit block cipher algorithm," 1994.

[2] L. Knudsen and B. Preneel, "Hash Functions Based on Block Ciphers and Quaternary Codes," Advances in Cryptology – Asiacrypt'96, Lecture Notes in Computer Sciences **1163**, Springer-Verlag, 1996, pp.77–90.

[3] R.L. Rivest, "The MD4 message digest algorithm," Advances in Cryptology – Crypto'90, Lecture Notes in Computer Sciences **537**, Springer-Verlag, 1991, pp.303–311.

[4] R.L. Rivest, "The MD5 message digest algorithm," Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, 1992.

[5] National Institute of Standards and Technology, "Secure Hash Standard," Federal Information Processing Standards, Publication **180-1**, 1995.

[6] H. Dobbertin, A. Bosselaers, B. Preneel, "RIPEMD-160: A Strengthened Version of RIPEMD," The Third Workshop of Fast Software Encryption, Lecture Notes in Computer Sciences **1039**, Springer-Verlag, 1996, pp.71–82.

[7] H. Dobbertin, "Cryptanalysis of MD4," The Third Workshop of Fast Software Encryption, Lecture Notes in Computer Sciences **1039**, Springer-Verlag, 1996, pp.53–69.