# その他評価が必要な暗号技術

# ハッシュ関数に関する調査

２００１年 1月

１． MD5

２． RIPEMD-160

３． SHA-1

# Abstract

Cryptographic hash functions have been designed in the first place to protect the authenticity of information.. As a practical cryptographic hash functions, dedicated hash functions have been proposed. Dedicated hash functions are those which are specifically designed from scratch for purpose of hashing, with optimized performance in mind. Those having received the greatest attention in practice are based on the MD4 hash function. MD4 was designed specifically for software implementation on 32-bits machines. Security concerns motivated the designed of MD5 shortly thereafter, as a more conservative variation of MD4. Other important subsequent variants include the SHA-1 and the hash function RIPEMD-160. However, the evaluation of MD4 based hash functions is difficult because the hash functions are designed by "trial-and-error" procedure. In general, the resistance of a particular hash function to known general attacks provides a measure of security. Accordingly, for the evaluation of the security of MD5, RIPEMD-160 and SHA-1, the attacks against them should be investigated. Therefore some attacks against MD4 based hash functions are reviewed in this report. This report shows that some attacks against MD5, RIPEMD-160, and SHA-1 do not yet threaten practical applications of them as well.

# １．はじめに

今日暗号学的一方向性ハッシュ関数（以下，単にハッシュ関数という）は現代の暗号技術においては基本的な構成要素の１つとして必要不可欠な存在になっている．特にディジタル署名におけるハッシュ関数の役割は大きく,安全なディジタル署名の構成のためには安全なハッシュ関数が必要不可欠である．ここで,暗号分野で用いるハッシュ関数に要求される性質としては

（1）Preimage resistance
（2）2nd preimage resistance
（3）Collision resistance

の３条件が挙げられる．これらの要求条件を満たすべく今日まで様々なハッシュ関数が提案されて来たが, 現在幅広く使われているのは基本的に Merkel と Damgard の理論に基づいた繰り返しハッシュ関数（Iterated Hash Function）である．特に MD4 が提案されて以来,その構造に基づいた多くのハッシュ関数が提案され,また,解析されてきた.現在,安全であると考えられているハッシュ関数としては MD5,RIPEMD-160, SHA-1 が挙げられる.本報告書ではこの３つのハッシュ関数の構造とその安全性に関して述べる.

# ２．歴史的な背景

1990 年に Rivest により MD4 が提案されて以来, MD4 の設計に基づいた多くのハッシュ関数が発表されて来た．MD4 は３２ビットマシーンに適合するように設計されたハッシュ関数であるため, ソフトウェア実装しても高速である．次いで 1991 年には MD4 を改良した MD5 が提案された．さらにヨーロッパの RIPE(Race Integrity Primitive Evaluation) プロジェクトでは MD4・MD5 に対する評価を行い, それに基づいて MD4 の変形である RIPEMD が 1995 年に提案されている．また,翌 1996 年には RIPEMD を改良した RIPEMD-128/160 が Dobbertin, Bosselaers, Preneel により提案された．一方, 米国では 1993 年に NIST(National Institute of Standard and Technology) による FIPS PUB 180 において SHA(Secure Hash Algorithm)が発表され, 1995 年には SHA の改良である SHA-1 が発表された．SHA-1 は米国政府の標準ハッシュ関数として現在公認されている．これら MD4 に基づいたハッシュ関数は高速ではあるが, その設計の基準などがこれまで特に明確にされていないため,その安全性の評価を行うにあたっては非常に困難がつきまとう．一般にこれら MD4 に基づいたハッシュ関数の安全性を評価する際には全探索攻撃より効率的な攻撃が存在するか否かにより評価される．

# ３．MDx 族ハッシュ関数

MD4 に基づいて構成されたハッシュ関数を MDx 族ハッシュ関数と呼ぶ．MDx 族ハッシュ関数は入力関数，圧縮関数，出力関数で構成されている．

## ３．１．準備

### ３．１．１．入力メッセージの変換

MD４に基づいたハッシュ関数は入力メッセージをブロックごとに分け，整数に変換してハッシュ関数の入力として使う．この入力の変換は多少の違いはあるが，すべての MDx 族ハッシュ関数におけるプリプロセスとして使われるためここでまとめて説明する．入力メッセージは３２ビットの整数に変換された後，長さが５１２の倍数になるようパディング処理が行われる．

メッセージを３２ビットで表現するための変換方式として，Little-endian と Big-endian がある．今，バイト列$B_i$が与えられたとする．ここで i はメモリアドレスである．Little-endian 方式は変換式$W = 2^{24}B_4 + 2^{16}B_3 + 2^8 B_2 + B_1$により４バイトの列$B_1, B_2, B_3, B_4$を１つの３２ビット整数 W に変換する．Big-endian は$W = 2^{24}B_1 + 2^{16}B_2 + 2^8 B_3 + B_4$により３２ビット変換する方式である．

### ３．１．２．基本演算

メッセージをハッシュする部分を圧縮関数と呼ぶことにすると MDx 族ハッシュ関数の圧縮関数はブール関数，法$2^{32}$での加算，巡回シフト演算で構成される．以下，法 $2^{32}$での加算は＋，n ビット左巡回シフトは $\ll n$ と表現する．

## ３．２．MD5

MD5[R92b]は MD4 に対する攻撃[BB91]が発表された後, Rivest により 1991 年に提案されたハッシュ関数である．基本的に MD4[R92a]の構造に基づいているが，MD4 を次のように改良した物である．

（1）　ラウンド数を４ラウンドに増やした.
（2）　各ステップで異なる定数を用いた.
（3）　２ラウンド目のブール関数をより非線形性を持つように構成した.
（4）　ステップ関数で，前ステップで計算された変数を加えることで“Avalanche effect”を増やした.
（5）　２，３ラウンドにおける入力が互いの順番が変えた.
（6）　高速“Avalanche effect”のため，各ラウンドで巡回シフト量が最適化した.

### ３.２.１．入力

入力は Little-endian 方式により３２ビット整数に変換され，512 ビットブロックに分けられる．512 ビットのメッセージブロックを X＝(X[0],X[1], … , X[15])と表現すると各 X[i] $(0 \le i \le 15)$ は３２ビットであり，次に説明する圧縮関数のステップ関数への入力として使われる.

### ３.２.２． 圧縮関数

圧縮関数の計算には４つの連鎖変数（A,B,C,D）を用いる．(A,B,C,D)の初期値 $IV = (h_1, h_2, h_3, h_4)$ としては

$$h_1 = 0\text{x}67452301, \quad h_2 = 0\text{xefcdab89}, \quad h_3 = 0\text{x}98\text{badcfe}, \quad h_4 = 0\text{x}10325476$$

を用いる．また，MD5 で使われるブール関数は次のように定義されている.

$$\begin{aligned} f(x,y,z) &= (x \wedge y) \vee (\bar{x} \wedge z) \\ g(x,y,z) &= (x \wedge z) \vee (y \wedge \bar{z}) \\ h(x,y,z) &= x \oplus y \oplus z \\ i(x,y,z) &= y \oplus (x \vee \bar{z}) \end{aligned}$$

このブール関数と２．１．２で与えた演算記号を用いると MD5 の圧縮関数のステップ関数は次のように表される.

1**ラウンド**　$FF(A,B,C,D,X[i],s,K[j]):A=B+(A+f(B,C,D)+X[i]+K[j])^{<<s}$

2**ラウンド**　$GG(A,B,C,D,X[i],s,K[j]):A=B+(A+g(B,C,D)+X[i]+K[j])^{<<s}$

3**ラウンド**　$HH(A,B,C,D,X[i],s,K[j]):A=B+(A+i(B,C,D)+X[i]+K[j])^{<<s}$

4**ラウンド**　$II(A,B,C,D,X[i],s,K[j])\ :\ A=B+(A+i(B,C,D)+X[i]+K[j])^{<<s}$

ここで，$K[j]\ (1\le j\le 64)$ は $2^{32}\cdot abs(\sin(j))$ の整数部分とする．ただし，j は Radian 値である．X[0]~X[15]は表１で与えられる順番によりステップ関数に入力される．

**表 1メッセージの適用順番**

| Step | Round1 | Step | Round 2 | Step | Round 3 | Step | Round 4 |
|---|---|---|---|---|---|---|---|
| 1 | X[0] | 17 | X[1] | 33 | X[5] | 49 | X[0] |
| 2 | X[1] | 18 | X[6] | 34 | X[8] | 50 | X[7] |
| 3 | X[2] | 19 | X[11] | 35 | X[11] | 51 | X[14] |
| 4 | X[3] | 20 | X[0] | 36 | X[14] | 52 | X[5] |
| 5 | X[4] | 21 | X[5] | 37 | X[1] | 53 | X[12] |
| 6 | X[5] | 22 | X[10] | 38 | X[4] | 54 | X[3] |
| 7 | X[6] | 23 | X[15] | 39 | X[7] | 55 | X[10] |
| 8 | X[7] | 24 | X[4] | 40 | X[10] | 56 | X[1] |
| 9 | X[8] | 25 | X[9] | 41 | X[13] | 57 | X[8] |
| 10 | X[9] | 26 | X[14] | 42 | X[0] | 58 | X[15] |
| 11 | X[10] | 27 | X[3] | 43 | X[3] | 59 | X[6] |
| 12 | X[11] | 28 | X[8] | 44 | X[6] | 60 | X[13] |
| 13 | X[12] | 29 | X[13] | 45 | X[9] | 61 | X[4] |
| 14 | X[13] | 30 | X[2] | 46 | X[12] | 62 | X[11] |
| 15 | X[14] | 31 | X[7] | 47 | X[15] | 63 | X[2] |
| 16 | X[15] | 32 | X[12] | 48 | X[2] | 64 | X[9] |

また左巡回シフト量 s も以下のように各ステップごとに定義されている．

表 2 左巡回シフトの値

| Step | Round1 | Step | Round 2 | Step | Round 3 | Step | Round 4 |
|---|---|---|---|---|---|---|---|
| 1 | 7 | 17 | 5 | 33 | 4 | 49 | 6 |
| 2 | 12 | 18 | 9 | 34 | 11 | 50 | 10 |
| 3 | 17 | 19 | 14 | 35 | 16 | 51 | 15 |
| 4 | 22 | 20 | 20 | 36 | 23 | 52 | 21 |
| 5 | 7 | 21 | 5 | 37 | 4 | 53 | 6 |
| 6 | 12 | 22 | 9 | 38 | 11 | 54 | 10 |
| 7 | 17 | 23 | 14 | 39 | 16 | 55 | 15 |
| 8 | 22 | 24 | 20 | 40 | 23 | 56 | 21 |
| 9 | 7 | 25 | 5 | 41 | 4 | 57 | 6 |
| 10 | 12 | 26 | 9 | 42 | 11 | 58 | 10 |
| 11 | 17 | 27 | 14 | 43 | 16 | 59 | 15 |
| 12 | 22 | 28 | 20 | 44 | 23 | 60 | 21 |
| 13 | 7 | 29 | 5 | 45 | 4 | 61 | 6 |
| 14 | 12 | 30 | 9 | 46 | 11 | 62 | 10 |
| 15 | 17 | 31 | 14 | 47 | 16 | 63 | 15 |
| 16 | 22 | 32 | 20 | 48 | 23 | 64 | 21 |

### ３.２.３. 出力

MD5 は圧縮関数の最後のステップで求められた連鎖変数の値と最初の初期値 IV を加えた後, (A,B,C,D)の４つの変数を結合することで１２８ビットのハッシュ値を出力する.

## ３.３. RIPEMD-160

RIPEMD-160 は RIPEMD に対する攻撃[Dob97]に基づいて改良されたハッシュ関数である. RIPEMD-160 は右ラインと左ラインの２つのアルゴリズムを並列で走らせて, 任意長のメッセージから 160 ビットのハッシュ値を出力する[DBP96]. RIPEMD-160 は MD5 のように入力の順番と左巡回シフト量が右ラインと左ラインに対して定められている. ５つの連鎖変数(A,B,C,D,E)を用い, ５ラウンド８０ステップで構成されている.

### ３.３.１. 入力

入力メッセージは Little-endian 方式により３２ビット整数に変換され，５１２ビットのブロックに分けられる．１６個の３２ビット入力 X[0]~X[15]は定められた順番によって右ラインと左ラインに入力される．表３でその入力順番を示す.

**表 3 RIPEMD-160 のメッセージの適用順番**

| Step Round1 | | | Step Round 2 | | | Step Round 3 | | | Step Round4 | | | Step Round5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Left | Right | | Left | Right | | Left | Right | | Left | Right | | Left | Right |
| 1 | X[0] | X[5] | 17 | X[7] | X[6] | 33 | X[3] | X[15] | 49 | X[1] | X[8] | 65 | X[4] | X[12] |
| 2 | X[1] | X[14] | 18 | X[4] | X[11] | 34 | X[10] | X[5] | 50 | X[9] | X[6] | 66 | X[0] | X[15] |
| 3 | X[2] | X[7] | 19 | X[13] | X[3] | 35 | X[14] | X[1] | 51 | X[11] | X[4] | 67 | X[5] | X[10] |
| 4 | X[3] | X[0] | 20 | X[1] | X[7] | 36 | X[4] | X[3] | 52 | X[10] | X[1] | 68 | X[9] | X[4] |
| 5 | X[4] | X[9] | 21 | X[10] | X[0] | 37 | X[9] | X[7] | 53 | X[0] | X[3] | 69 | X[7] | X[1] |
| 6 | X[5] | X[2] | 22 | X[6] | X[13] | 38 | X[15] | X[14] | 54 | X[8] | X[11] | 70 | X[12] | X[5] |
| 7 | X[6] | X[11] | 23 | X[15] | X[5] | 39 | X[8] | X[6] | 55 | X[12] | X[15] | 71 | X[2] | X[8] |
| 8 | X[7] | X[4] | 24 | X[3] | X[10] | 40 | X[1] | X[9] | 56 | X[4] | X[0] | 72 | X[10] | X[7] |
| 9 | X[8] | X[13] | 25 | X[12] | X[14] | 41 | X[2] | X[11] | 57 | X[13] | X[5] | 73 | X[14] | X[6] |
| 10 | X[9] | X[6] | 26 | X[0] | X[15] | 42 | X[7] | X[8] | 58 | X[3] | X[12] | 74 | X[1] | X[2] |
| 11 | X[10] | X[15] | 27 | X[9] | X[8] | 43 | X[0] | X[12] | 59 | X[7] | X[2] | 75 | X[3] | X[13] |
| 12 | X[11] | X[8] | 28 | X[5] | X[12] | 44 | X[6] | X[2] | 60 | X[15] | X[13] | 76 | X[8] | X[14] |
| 13 | X[12] | X[1] | 29 | X[2] | X[4] | 45 | X[13] | X[10] | 61 | X[14] | X[9] | 77 | X[11] | X[0] |
| 14 | X[13] | X[10] | 30 | X[14] | X[9] | 46 | X[11] | X[0] | 62 | X[5] | X[7] | 78 | X[6] | X[3] |
| 15 | X[14] | X[3] | 31 | X[11] | X[1] | 47 | X[5] | X[4] | 63 | X[6] | X[10] | 79 | X[15] | X[9] |
| 16 | X[15] | X[12] | 32 | X[8] | X[2] | 48 | X[12] | X[13] | 64 | X[2] | X[14] | 80 | X[13] | X[11] |

### ３.３.２． 圧縮関数

圧縮関数の計算には５つの連鎖変数（A,B,C,D,E）を用いる．A,B,C,D の初期値は MD5 と同じ値であり，新しく E の初期値が定められている．(A,B,C,D,E)の初期値

$IV = (h_1, h_2, h_3, h_4, h_5)$ を以下に示す.

$h_1 = 0\text{x}67452301,\ h_2 = 0\text{x}efcdab89,\ h_3 = 0\text{x}98badcfe,\ h_4 = 0\text{x}10325476,\ h_5 = 0\text{x}c3d2e1f0$

この初期値は左右両ラインで共通に用いられる．また，圧縮関数では次の５つのブール関数を用いる．

$$
\begin{aligned}
f(x,y,z) &= x \oplus y \oplus z \\
g(x,y,z) &= (x \wedge y) \vee (\bar{x} \wedge z) \\
h(x,y,z) &= (x \vee \bar{y}) \oplus z \\
k(x,y,z) &= (x \wedge z) \vee (y \wedge \bar{z}) \\
l(x,y,z) &= x \oplus (y \vee \bar{z})
\end{aligned}
$$

RIPEMD-160 の圧縮関数を構成するステップ関数を以下に示す．ただし，$Z_R(\cdot)$，$Z_L(\cdot)$でそれぞれ右ライン, は左ラインの関数･変数･定数などを表す. RIPEMD-160 は右ラインと左ラインを並列に実行することでハッシュを行う．ステップ関数で用いられる定数$K_L[j]$，$K_R[j]$は次のように与えられる．

| | | |
|---|---|---|
| $K_L[j] = 0,$ | $K_R[j] = 0x50a28be6$ | $(1 \le j \le 16)$ |
| $K_L[j] = 0x5a27999$ | $K_R[j] = 0x5c4dd124$ | $(17 \le j \le 32)$ |
| $K_L[j] = 0x6ed9eba1$ | $K_R[j] = 0x6d703ef3$ | $(33 \le j \le 48)$ |
| $K_L[j] = 0x8f1bbcdc$ | $K_R[j] = 0x7a6d76e9$ | $(49 \le j \le 64)$ |
| $K_L[j] = 0xa953fd4e$ | $K_R[j] = 0$ | $(65 \le j \le 80)$ |

また，ステップ関数で用いられる左巡回シフト量$s_L[j]$，$s_L[j]$は表４のように与えられる．

表 4 左巡回シフトの値

| Step Round1 | | | Step Round 2 | | | Step Round 3 | | | Step Round4 | | | Step Round5 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Left | Right | | Left | Right | | Left | Right | | Left | Right | | Left | Right |
| 1 | 11 | 8 | 17 | 7 | 9 | 33 | 11 | 9 | 49 | 11 | 15 | 65 | 9 | 8 |
| 2 | 14 | 9 | 18 | 6 | 13 | 34 | 13 | 7 | 50 | 12 | 5 | 66 | 15 | 5 |
| 3 | 15 | 9 | 19 | 8 | 15 | 35 | 6 | 15 | 51 | 14 | 8 | 67 | 5 | 12 |
| 4 | 12 | 11 | 20 | 13 | 7 | 36 | 7 | 11 | 52 | 15 | 11 | 68 | 11 | 9 |
| 5 | 5 | 13 | 21 | 11 | 12 | 37 | 14 | 8 | 53 | 14 | 14 | 69 | 6 | 12 |
| 6 | 8 | 15 | 22 | 9 | 8 | 38 | 9 | 6 | 54 | 15 | 14 | 70 | 8 | 5 |
| 7 | 7 | 15 | 23 | 7 | 9 | 39 | 13 | 6 | 55 | 9 | 6 | 71 | 13 | 14 |
| 8 | 9 | 5 | 24 | 15 | 11 | 40 | 15 | 14 | 56 | 8 | 14 | 72 | 12 | 6 |
| 9 | 11 | 7 | 25 | 7 | 7 | 41 | 14 | 12 | 57 | 9 | 6 | 73 | 5 | 8 |
| 10 | 13 | 7 | 26 | 12 | 7 | 42 | 8 | 13 | 58 | 14 | 9 | 74 | 12 | 13 |
| 11 | 14 | 8 | 27 | 15 | 12 | 43 | 13 | 5 | 59 | 5 | 12 | 75 | 13 | 6 |
| 12 | 15 | 11 | 28 | 9 | 7 | 44 | 6 | 14 | 60 | 6 | 9 | 76 | 14 | 5 |
| 13 | 6 | 14 | 29 | 11 | 6 | 45 | 5 | 13 | 61 | 8 | 12 | 77 | 11 | 15 |
| 14 | 7 | 14 | 30 | 7 | 15 | 46 | 12 | 13 | 62 | 6 | 5 | 78 | 8 | 13 |
| 15 | 9 | 12 | 31 | 13 | 13 | 47 | 7 | 7 | 63 | 5 | 15 | 79 | 5 | 11 |
| 16 | 8 | 6 | 32 | 12 | 11 | 48 | 5 | 5 | 64 | 12 | 8 | 80 | 6 | 11 |

RIPEMD-160 のステップ関数は次の通りである.

1 **ラウンド**($1 \le j \le 16$)

$FF_L(A_L, B_L, C_L, D_L, E_L, X[i], s_L[j], K_L[j])$:

$$A_L = (A_L + f(B_L, C_L, D_L) + X[i] + K_L[j])^{<<s_L[j]} + E_L, \quad C_L = C_L^{<<10}$$

$LL_R(A_R, B_R, C_R, D_R, E_R, X[i], s_R[j], K_R[j])$:

$$A_R = (A_R + l(B_R, C_R, D_R) + X[i] + K_R[j])^{<<s_R[j]} + E_R, \quad C_R = C_R^{<<10}$$

2 **ラウンド**($17 \le j \le 32$)

$GG_L(A_L, B_L, C_L, D_L, E_L, X[i], s_L[j], K_L[j])$:

$$A_L = (A_L + g(B_L, C_L, D_L) + X[i] + K_L[j])^{<<s_L[j]} + E_L, \quad C_L = C_L^{<<10}$$

$KK_R(A_R, B_R, C_R, D_R, E_R, X[i], s_R[j], K_R[j])$:

$$A_R = (A_R + k(B_R, C_R, D_R) + X[i] + K_R[j])^{<<s_R[j]} + E_R, \quad C_R = C_R^{<<10}$$

3 **ラウンド**($33 \le j < 48$)

$HH_L(A_L, B_L, C_L, D_L, E_L, X[i], s_L[j], K_L[j])$:

$$A_L = (A_L + h(B_L, C_L, D_L) + X[i] + K_L[j])^{<<s_L[j]} + E_L, \quad C_L = C_L^{<<10}$$

$HH_R(A_R, B_R, C_R, D_R, E_R, X[i], s_R[j], K_R[j])$:

$$A_R = (A_R + h(B_R, C_R, D_R) + X[i] + K_R[j])^{<<s_R[j]} + E_R, \quad C_R = C_R^{<<10}$$

4 **ラウンド**($49 \le j \le 64$)

$KK_L(A_L, B_L, C_L, D_L, E_L, X[i], s_L[j], K_L[j])$:

$$A_L = (A_L + k(B_L, C_L, D_L) + X[i] + K_L[j])^{<<s_L[j]} + E_L, \quad C_L = C_L^{<<10}$$

$GG_R(A_R, B_R, C_R, D_R, E_R, X[i], s_R[j], K_R[j])$:

$$A_R = (A_R + g(B_R, C_R, D_R) + X[i] + K_R[j])^{<<s_R[j]} + E_R, \quad C_R = C_R^{<<10}$$

5**ラウンド**($65 \le j \le 80$)

$LL_L(A_L, B_L, C_L, D_L, E_L, X[i], s_L[j], K_L[j])$:

$$A_L = (A_L + l(B_L, C_L, D_L) + X[i] + K_L[j])^{<<s_L[j]} + E_L, \quad C_L = C_L^{<<10}$$

$FF_R(A_R, B_R, C_R, D_R, E_R, X[i], s_R[j], K_R[j])$:

$$A_R = (A_R + f(B_R, C_R, D_R) + X[i] + K_R[j])^{<<s_R[j]} + E_R, \quad C_R = C_R^{<<10}$$

### ３.３.３． 出力

出力は基本的に MD5 同様, 最後の段階で求められた連鎖変数の値と初期値 IV を加えた後, (A,B,C,D,E)の 5 つの変数を結合することでハッシュ値を出力するが, ２つのラインを使うため少し違う形で計算される：

$$A = h_2 + C_L + D_R, \quad B = h_3 + D_L + E_R, \quad C = h_4 + E_L + A_R,$$
$$D = h_5 + A_L + B_R, \quad E = h_1 + B_L + C_R.$$

## ３.４. SHA-1

SHA-1 は NIST（National Institute of Standard and Technology）により提案された SHA（Secure Hash Algorithm）を改良したハッシュ関数である[N95]. SHA-1 の特徴は入力メッセージをブロック暗号における鍵のように考えて使うところにある. すなわち, 入力メッセージを用いて各ステップで新しいメッセージを生成し, それを入力としてステップ関数を作用させる. この新しいメッセージの生成はメッセージ操作に基づく攻撃に耐性を持つと考えられる. SHA-1 は各メッセージブロックに対し４ラウンド８０ステップの演算を行い, １６０ビットのハッシュ値を出力する.

### ３.４.１． 入力

入力メッセージは Big-endian 方式により３２ビットに変換され, ５１２ビットのブロックに分けられる. メッセージブロック X＝(X[0],X[1], … , X[15])を用いて次の式

$$X[j] = (X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16])^{<<1} \qquad (16 \le j \le 79)$$

により新しい X[16],X[17], … , X[79]を生成する. したがって, (X[0],X[1], …, X[79]) が圧縮関数の入力として用いられる.

### ３.４.２. 圧縮関数

圧縮関数の計算には５つの連鎖変数（A,B,C,D,E）を用いる. A,B,C,D の初期値は MD5 と同じ値であり, 新しく E の初期値が定められている. (A,B,C,D,E)の初期値 $IV = (h_1, h_2, h_3, h_4, h_5)$ を以下に示す.

$h_1 = 0\text{x}67452301,\ \ h_2 = 0\text{x}efcdab89,\ \ h_3 = 0\text{x}98badcfe,\ \ h_4 = 0\text{x}10325476,\ \ h_5 = 0\text{x}c3d2e1f0$

SHA-1 の圧縮関数で使われるブール関数はステップｔに対し次のように定義されている.

$$
\begin{aligned}
f^{(t)}(x,y,z) &= (x \wedge y) \vee (\bar{x} \wedge z) && (1 \le t \le 20)\\
f^{(t)}(x,y,z) &= x \oplus y \oplus z && (21 \le t \le 40)\\
f^{(t)}(x,y,z) &= (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) && (41 \le t \le 60)\\
f^{(t)}(x,y,z) &= x \oplus y \oplus z && (61 \le t \le 80)
\end{aligned}
$$

ステップ関数は次のように設計されている.

$$A = A^{<<5} + f^{(t)}(B,C,D) + E + X[j] + K^{(t)};\quad E = D;\quad D = C;\quad C = B^{<<30};\quad B = A;$$

ただし，$1 \le t \le 80$，$0 \le j \le 79$ である．また，定数 $K^{(t)}$ の値としては次の値を用いる.

$$
\begin{aligned}
K^{(t)} &= 0x5a827999 && (1 \le t \le 20)\\
K^{(t)} &= 0x6ed9eba1 && (21 \le t \le 40)\\
K^{(t)} &= 0x8f1bbcdc && (41 \le t \le 60)\\
K^{(t)} &= 0xca62c1d6 && (61 \le t \le 80)
\end{aligned}
$$

### ３.４.３． 出力

MD5 同様圧縮関数で求められた値と初期値 IV を加算した後，５つの変数（A,B,C,D,E）の結合により１６０ビットのハッシュ値を出力する.

## ３.５. MD5, RIPEMD-160, SHA-1 の構造の比較

MD5, RIPEMD-160, SHA-1 の圧縮関数に用いられている各構成要素をラウンド数・ブール関数・入力の仕方に分けて比較し,まとめる.

表 5 用いられるブール関数の種類

| ブール関数の種類 | MD5 | RIPEMD-160 | SHA-1 |
|---|---|---|---|
| $f(x,y,z)=(x\wedge y)\vee(\bar{x}\wedge z)$ | ○ | ○ | ○ |
| $g(x,y,z)=(x\wedge z)\vee(y\wedge\bar{z})$ | ○ | ○ | $(x\wedge y)\vee(x\wedge z)\vee(y\wedge z)$ * |
| $h(x,y,z)=x\oplus y\oplus z$ | ○ | ○ | ○ |
| $i(x,y,z)=y\oplus(x\vee\bar{z})$ | ○ | $x\oplus(y\vee\bar{z})$ ** | -- |
| $j(x,y,z)=(x\vee\bar{y})\oplus z$ | -- | ○ | -- |

( ○ :ブール関数が用いられている.

-- ： ブール関数が用いられていない.

* ：MD5 の g(x,y,z)の代わりに用いられるブール関数.

* * : MD5 の i(x,y,z)の代わりに用いられるブール関数.)

表 6 ハッシュ関数の各パラメータの比較

| | MD5 | RIPEMD-160 | SHA-1 |
|---|---|---|---|
| ラウンド数<br>(ステップ数) | 5<br>(64) | 5×2 *<br>(80×2) * | 4<br>(80) |
| 入力ブロックの<br>サイズ(bit) | 512 | 512 | 512 |
| 出力サイズ(bit) | 128 | 160 | 160 |
| 入力の変換方式 | Little-endian | Little-endian | Big-endian |
| 各ステップでの<br>入力の仕方 | 表１で定義された<br>順に入力 | 表３で定義された<br>順に入力 | メッセージを拡張 |
| 左巡回シフト量 | 表２で定義 | 表４とステップ<br>関数内で定義 | ステップ関数内で<br>定義 |

(*：右ラインと左ラインの両方でステップ関数を行うため,実際は２倍のラウンド数を必要とする.)

# ４．安全性評価

一般に，暗号系は適当な仮定の下でその安全性が証明されていることが望ましいが，（実用的な）ハッシュ関数の場合は共通鍵暗号の場合と同様その安全性を理論的に証明することは困難である．したがってハッシュ関数の安全性は，既存の攻撃に対して耐性を持つか否かによって評価されている．そこで本章では，ハッシュ関数に対する体表的な攻撃について述べた後，MD５，SHA-1, RIPEMD-160 に対する攻撃を紹介し，それらの攻撃に対する各ハッシュ関数の強度評価について述べる．

## ４.１．ハッシュ関数に対する主な攻撃

すでに述べたように，安全なハッシュ関数はその Preimage や 2nd preimage，さらに衝突(Collision)を求めることが困難でなければならない．したがって，もしこれらを実際に求める方法を与えることができれば，それがハッシュ関数に対する攻撃となる．標準的なハッシュ関数に対する攻撃としては，

（１） アルゴリズムと独立の攻撃
（２） 圧縮関数に対する攻撃
（３） 連鎖に基づいた攻撃(Chaining attacks)

が挙げられる．まずこの３つの攻撃について簡単に説明する．

### ４.１.１．アルゴリズムと独立の攻撃

以下で説明する攻撃は，ハッシュ関数（アルゴリズム）の構造にはよらない素朴な全数探索的な攻撃である．これらの攻撃において各ハッシュ関数は，出力ビット長および１回の演算に要する時間のみによって特徴付けられるブラックボックスとして扱われる．したがって，これらはすべてのハッシュ関数に対して適用可能であり，それゆえハッシュ関数の安全性の基準を与える．すなわち，ハッシュ関数に対するある攻撃が，以下に述べる攻撃と同程度の計算を必要とする場合，ハッシュ関数はその攻撃に対して安全であるとみなされる．

（１）ランダム攻撃
ランダム攻撃は，Preimage,や 2nd preimage を全数探索により求める攻撃である．（衝突は，以下で述べるバースデー攻撃によりさらに効率的に求めることができる．）攻撃者は，あるメッセージ $X$ のハッシュ値 $H(X)$ に対し， 別のメッセージ $\tilde{X}$

をランダムに選び，そのハッシュ値が $H(X) = H(\tilde{X})$ となることを期待する．ハッシュ値を一様な n ビットのランダム変数と仮定した場合，この攻撃は $2^{-n}$ の成功確率を持つ.

（2）バースデー攻撃

バースデー攻撃は，バースデーパラドクスを利用して全数探索よりも効率的に衝突(Collision)を求める攻撃である．ここでハッシュ値を一様なランダム変数と仮定した場合，衝突発見のためには $O(2^{n/2})$ の計算が必要である．この攻撃はアルゴリズムと独立な攻撃であるため，これより効率的な（衝突を求める）攻撃が見つからない場合，ハッシュ関数は衝突困難であるとみなされる.

### ４.１.２． 圧縮関数に対する攻撃

圧縮関数に対する攻撃の目標は，あるステップ i で preimage，pseudo-preimage，2nd-preimage, collision, pseudo-collision を探索する方法を与えることである. 圧縮関数に対する攻撃が重要な理由は，多くの場合，ハッシュ関数の全体の安全性は，その圧縮関数の安全性に強く依存するからである．以下では，ハッシュ関数の圧縮関数を H(・)，異なる２つのメッセージを $X$ および $\tilde{X}$，ある与えられたハッシュ値を Y とあらわすことにする．また，$V$ と $\tilde{V}$ はそれぞれ $X$ と $\tilde{X}$ を入力とする圧縮関数で用いられる初期値 IV の適当な値とする．$X_0$，$V_0$ はそれぞれある与えられた入力，初期値とする.

（1）Preimage

Preimage を発見するということは

$$H(V_0, X) = Y$$

となる X を探すことである．すなわち，ハッシュ値が与えられた時，そのハッシュ値からメッセージを探し出すことである.

（2）Pseudo-preimage

Pseudo-preimage の探索とは

$$H(V, X) = Y$$

を満たす V と X を計算することである. すなわち，与えられたハッシュ値からランダムな初期値と入力メッセージを計算する攻撃である.

（3）2nd preimage

2nd preimage の探索とは

$$H(V_0, \tilde{X}) = H(V_0, X_0)$$

となる $\tilde{X}$ を計算することである．ある入力メッセージが与えられた時，それと同じハッシュ値を持つ異なるメッセージを探す攻撃である．

（4）衝突（Collision）

一般に衝突とは，異なる２つのメッセージが同じハッシュ値を持つことである．衝突には固定された IV に対する衝突と適当な IV に対する衝突が考えられる．固定 IV に対する衝突とは

$$H(V_0, X) = H(V_0, \tilde{X})$$

を満たす $X$ と $\tilde{X}$ を探すことである．すなわち，圧縮関数のアルゴリズムの中で与えられた初期値を用いて２つの異なるメッセージ $X$ と $\tilde{X}$ を探すことである．一方，ランダムな IV に対する衝突とは

$$H(V, X) = H(V, \tilde{X})$$

を満たす $X$ ， $\tilde{X}$ ，V を計算することである．これは攻撃者によりランダムに選ばれた初期値を用いて異なる２つのメッセージ $X$ と $\tilde{X}$ を計算することである．通常，単に衝突といえば前者を指す．

（5）擬似衝突（Pseudo-collision）

擬似衝突とは

$$H(V, X) = H(\tilde{V}, \tilde{X})$$

を満たす $X$ ， $\tilde{X}$ ， $V$ ， $\tilde{V}$ を計算することである．すなわち，同じハッシュ値を出力する異なる２つの初期値と入力の対を求める攻撃である．

### ４．１．３． 連鎖に基づいた攻撃(Chaining attacks)

ここでは連鎖変数を用いる繰り返しハッシュ関数の特徴に基づく攻撃について述べる．メッセージはm個のブロックに分けられてハッシュ関数に入力されるとする．m個のブロックの中から１つを選んで別の値のブロックに替えても同じハッシュ

値を出力するようなブロックを発見することが連鎖に基づいた攻撃の目標である．したがって，連鎖に基づいた攻撃の焦点は，ハッシュ関数全体ではなく，むしろその圧縮関数にある．以下に，代表的な連鎖攻撃をいくつか挙げる．

（1） Correcting-block chaining attacks

与えられたメッセージに対して，その値を別の値に置き換えてもハッシュ値が変わらないようなブロックを探索する攻撃である．この攻撃は Preimage や Collision の発見に使われる．

（2） Meet-in-the-middle chaining attacks

ハッシュ関数の途中の値に対する衝突を探す方式であり，与えられたハッシュ値に対するメッセージを探索する攻撃である．攻撃者は，ある位置でメッセージブロックをアルゴリズムで定義されている初期値により前方向にハッシュし，与えられたハッシュ値により後方向にハッシュしながらその位置における連鎖変数の衝突をさがす．この攻撃を行うために攻撃者は連鎖において後方向にも効率よく計算できる必要がある．

（3） Fixed-point chaining attacks

メッセージに任意のブロックを挿入しても全体のハッシュ値が変わらない位置を発見することにより，ハッシュ関数に対する 2nd preimage や collision を検索する攻撃である．

（4） Differential chaining attacks

差分攻撃はブロック暗号において効率のよい攻撃法として知られているが，同時にハッシュ関数においても強力な攻撃となる．この攻撃は入力の差分と出力の差分を比較することにより行われる．特に，ある入力差分に対し出力差分が０になると衝突が起こるものとする．

## ４．２．MDx 族ハッシュ関数への攻撃

MDx 族ハッシュ関数に対する攻撃は主に圧縮関数に関する攻撃である．特にハッシュ関数のアルゴリズムの中で用いられる初期値に対する衝突の探索が実用的な攻撃として重要である．MDx 族ハッシュ関数に対する攻撃には 4.1.2 の圧縮関数に対する攻撃と 4.1.3 の連鎖に基づいた攻撃により行われる．

### ４．２．１．MD4 に対する攻撃

（1）２－３ラウンドに対する攻撃

Boer と Bosselaers により提案された MD4 に対する攻撃であり，仮に計算が圧縮関数の第１ラウンドを省略した２－３ラウンドに対する攻撃である[BB91]．２ラウンドの５ステップから１３ステップに入力される X[1], X[5], X[9], X[13], X[6], X[10]は３ラウンドのステップ２１からステップ２９の間でも異なる順番ではあるが，入力として用いられることに着目し，同じハッシュ値を出力する入力と連鎖変数の値を探索する攻撃である．衝突の探索のために計算しなければならない未知数は

① $(A_5, B_5, C_5, D_5)$ の値

② $(A_9, B_9, C_9, D_9)$ に対する２つの値

③ $(A_{13}, B_{13}, C_{13}, D_{13})$ の値

④ $(A_{21}, B_{21}, C_{21}, D_{21})$ の値

⑤ $(A_{25}, B_{25}, C_{25}, D_{25})$ に対する２つの値

⑥ $(A_{29}, B_{29}, C_{29}, D_{29})$ の値

⑦ X[2],X[6],X[10],X[14],X[1],X[5],X[9],X[13]に対する２つの値

の４８個である．この未知数を計算するために

(a) (A,B,C,D)の最終的な値

(b) $(A_{17}, B_{17}, C_{17}, D_{17})$の値

(c) X[0], X[4], X[8], X[12], X[3], X[7], X[11], X[15]の値

の１６個の未知数を持つ１６個の式を考える．また，N＝0x55555555 と定め，N の奇数の位置は０，偶数の位置は１とする．N を奇数ビット巡回シフトするとその値は２N になるという特徴を用いて②と⑤を計算し，その値と一致する初期値（A,B,C,D）を探す．次に１６個の与えられた式を満たす入力として２つの X[1],X[2],X[5],X[10],X[13],X[14]を選んだ後，X[6],X[9]を選ぶ.
このプロセスにより求められた２つのメッセージ間の差分はいつも

0 -2N 2N 0 0 -2N 2N 0 0 -N N 0 0 -N N 0

となる．この攻撃は１６MHz IBM PS/2 で衝突が発見するのに１ms を要することが報告されている.

（２）Almost Collision
Vaudenay は MD4 の圧縮関数の構造と入力順番を考慮すると Almost Collision を発見できることを示した[V95]．この攻撃も（１）と同様入力が単純であることを利用した攻撃である．X[15]の値を変えて１，２ラウンドで用いると２ラウンドの終わりでは連鎖変数(A,B,C,D)の中で B の値だけが変わること，すなわち，X[15]の値を変えることで一部だけ値の異なるハッシュ値が得られることを示した．Almost Collision の存在は Collision の探索に有用であると考えられ，実際に MD4 の全段階における攻撃[Dob96a]と MD5 に対する攻撃[Dob96b]にも利用されている.

（３）全ラウンドに対する攻撃
Dobbertin により提案された MD4 の全段階に対する攻撃が１９９６年に発表された[Dob96a]．連鎖変数に基づいた攻撃や Almost Collision，差分攻撃などの様々な攻撃法を用いて構成されている． この攻撃は

① ステップ１３－２０間の Almost Collision
② ステップ２１－３６間の法$2^{32}$での差分攻撃
③ ステップ１－１２間での正しい初期値に対する衝突

の３つの部分に構成されている.

X[12]を選びステップ１３－２０間の Almost Collision を探索するため，ステップ２０での出力差分$\Delta_{20}$が

$$\Delta_{20} = (0,1^{<<25},-1^{<<5},0)$$

となる $X$ と $\tilde{X}$ を探す．$X$ と $\tilde{X}$ はステップ１３から２０の間，差分が$\Delta_{20}$である Almost Collision である．次に $X$ =(X[0],X[1],…,X[15])と $\tilde{X} = (\tilde{X}[0], \tilde{X}[1],..., \tilde{X}[15])$ を

$$X[i] = \tilde{X}[i] \quad (i \neq 12)$$
$$X[j] = \tilde{X}[j] + 1 \quad (j = 12)$$

とし，ステップ３６での出力差分$\Delta_{36}$が０になるような $X$ と $\tilde{X}$ を計算する．

**[ 定理１]** Almost Collision の条件を満たすようにステップ１３からステップ２０までの入力を探した後，他の入力を探すためにランダムに X[i]を選び，ステップ１２からステップ１まで後方向に圧縮を行う．ここで

$$(A_{12}, B_{12}, C_{12}, D_{12}) = (A, B, C, D)$$

とすると $\Delta_{36} = 0$ になる $X$ と $\tilde{X}$ を探す確率は $2^{-22}$ である．

差分攻撃の後，連鎖変数(A,B,C,D)の初期値としてアルゴリズムで与えられた値を用いて，ステップ１から１２まで用いられている入力の中で一部を直す．
Dobbertin はこの攻撃で実際は $2^{22}$ の計算量以内で衝突が発見出来ることを示した．

### ４．２．２． MD5 に対する攻撃

MD4 の２－３ラウンドに対する攻撃が発表された後，その攻撃から分かった弱点を改良したハッシュ関数として提案されたのが MD5 である．その改良点は MD5 のアルゴリズムのところで述べている．しかし，この改良にも関わらず MD5 に対する攻撃はいくつか発表されている．

（１）１－２ラウンドに対する攻撃

Boer と Bosselaers により提案された MD5 に対する攻撃は，連鎖変数（A,B,C,D)の初期値 $IV = (h_1, h_2, h_3, h_4)$ と４ビット異なる値を初期値として用いることで擬似

衝突を探索する攻撃である[BB93].
この攻撃のため，まず，MSB が１になる（A,B,C,D)をランダムに生成する．圧縮関数の各ステップで連鎖変数の値が F8000000 になるよう X＝(X[0],X[1], … , X[15])を選ぶ．選ばれた X が１，２ラウンドにおいて有効な選択であるかを MSB の値が１であるかで判断する．１ラウンドのステップｉでＭＳＢが１でない場合，X[i-1] をステップ i での巡回シフト量によりシフトした値が採用される（巡回シフト量は表２を参照）．この値を２ラウンドで用い，他のステップの MSB に変化が生じるかを調べる．変化が生じた場合，各ステップで計算される変数の値が F8000000 に近似するようにメッセージの他の部分を変える．
Boer と Bosselaers は圧縮関数に対する擬似衝突を $2^{16}$ の操作で発見できることを示した．この攻撃は擬似衝突(Pseudo-Collision)を与えるが，連鎖変数の固定された初期値に対する衝突の発見まで拡張できないため，実用的な攻撃とはみなされていない．

（２）MD5 の全ラウンドに対する攻撃
１９９６年 MD5 に関する衝突例が Dobbertin により発表された[Dob96b]．Dobbertin の攻撃は初期値を適当に選ぶことによって X[14]における衝突を与える．

まず $X[i]$ と $\tilde{X}[i]$ だけが異なり，他のすべてが同一のメッセージ $X$ と $\tilde{X}$ の対を考える．小さなハミング重み $\Delta$ を選び，次式が成り立つとする．

$$\tilde{X}[i] = X[i] + \Delta$$

ここで特に i=14 の場合を考える．この時表１により $X[14]$ と $\tilde{X}[14]$ は１ラウンドの１５ステップ，２ラウンドの２６ステップ，３ラウンドの３６ステップ，４ラウンドの５１ステップで使われることがわかる．攻撃のため注意して考察しなければならない部分は

(a) １５ステップから２６ステップの間と
(b) ３６ステップから５１ステップの間

の演算である．$X$ と $\tilde{X}$ に対する圧縮関数は１４ステップまで一緒である．$X[14]$ と $\tilde{X}[14]$ が入力として用いられるステップ１５，すなわち，（a）の始まり部分で最初

の誤差が起こる．もし（a）に対して $X[14]$ と $\tilde{X}[14]$ が２回目に使われるステップ２６で連鎖変数の値を一致させることが出来れば，これを部分（a）の内部衝突と呼ぶ．（a）以降から $X[14]$ と $\tilde{X}[14]$ は用いられないので $X$ と $\tilde{X}$ に対する圧縮関数はステップ２７から３５まではまた同じである．その後，２つの入力 $X$ と $\tilde{X}$ に対し圧縮関数が同じ出力をもつために部分（b）の内部衝突を必要とする．Dobbertin はこの方式で Pentium PC で１０時間ぐらいで衝突を発見できることを示している．彼は MD5 に対する完全な攻撃は

① ラウンド１－２における内部衝突
② ラウンド３－４における内部衝突
③ これら２つの内部衝突を結ぶ

の３部分で構成されると主張している．ステップ関数の定義に基づいて内部衝突を方程式系で表すことで①と②を解決できると考えられる．しかし，MD５の完全な解読のためにはさらに複雑な演算が必要されると考えられている．

### ４．２．３． RIPEMD-160 に対する攻撃

RIPEMD-160 の前のバージョンである RIPEMD の１ラウンドまたは３ラウンドを省略した圧縮関数に対する攻撃は

(a) 内部衝突
(b) 後方向衝突
(c) RIPEMD の初期値による衝突

の３つの部分で構成されている．内部衝突の探索のため，

$$X[i_0] \neq \tilde{X}[i_0],$$
$$X[i] = \tilde{X}[i] \quad for\ i \neq i_0$$

となる異なる２つのメッセージ $X$ ，$\tilde{X}$ を探索する．この内部衝突が見つかると２ラウンドでの衝突を探索するため，$X[i_0]$ が用いられるステップ以外で使われる連鎖変数の値をステップ $(i_0 - 1)$ 以降では同じ値になるように決める．この部分を後方

向衝突と呼ぶ．後方向衝突は，ある位置を中心に前後方向でステップ関数を作用させるので Meet-in-the-middle attack を用いている．
後方向衝突の段階ではランダムに生成される初期値を用いるが，それをアルゴリズムで与えられた値に固定して衝突を探索する．この攻撃によると最初のラウンドまたは最後のラウンドを省略した時，$2^{31}$ 以内の計算で衝突を発見することができることが報告されている[Dob97]．この攻撃は左右２つのラインの入力パタンーや左巡回シフト量などが同じであるため可能だと考えられる.この攻撃が可能なのは RIPEMD の左右２つのラインのプロセスが定数以外はまったく同一であるためである.したがってこの攻撃に対して耐性を持たせるためには２つのラインのプロセスを独立にする必要がある.RIPEMD に対して左右ラインのプロセスを違いに独立にし,さらにラウンド数を５に拡張することにより安全性を高めたのが RIPEMD-160 である.RIPEMD から RIPEMD-160 への改良点を以下に具体的に挙げる.

（１）ラウンド数を３ラウンドから５ラウンドに増やした.
（２）並列の左右ラインアルゴリズムを独立に設計した.
（３）左右ライン異なるメッセージの入力順番を用いた ： １－２ラウンドで近い位置にあった入力は２－３ラウンドでは離れるように設計されている．また,左ラインで隣接している２つの入力は右ラインでは少なくとも７距離れるようにしている.
（４）MD５と同じブール関数を用いた.
（５）左右ラインで用いられるブール関数の適用順番を逆にした.

また,安全性向上のため,左巡回シフト量の決定において以下の法則を用いている.

(a) シフト量は５から１５間の値とする.
(b) 各メッセージブロックは異なる量でシフトされ,異なるパリティを持つようにする.
(c) 各連鎖変数に用いられるシフトは特別なパタンーを持たない.
(d) 多くのシフト定数が４により分けられては行けない.

これらは RIPEMD,MD4 に対する攻撃を考慮した改良であるため,MD4, RIPEMD に対する攻撃は RIPEMD-160 には適用できず，実際,バースデー攻撃より効率的な RIPEMD-160 に対する攻撃はまだ報告されていない.

### ４.２.４. SHA-1 に対する攻撃

MD5 や RIPEMD-160 がその前のバージョンに対する攻撃に基づき，改良されたことに比べ，以前のバージョンの攻撃が発表する前に改良されたハッシュ関数が SHA-1 である．SHA-1 のアルゴリズムで示したように SHA も SHA-1 も圧縮関数の解析のためにはステップ関数だけでなく，入力メッセージを拡張する部分も分析しなければならない．そのため，MD4，MD5，RIPEMD への攻撃は SHA,SHA-1 には適用できない.

SHA に対する攻撃は1998年に Chabaud と Joux によりはじめて報告された[CJ98]. この攻撃は出力差分が０になるメッセージ $X$ と $\tilde{X}$ を探すため, 入力に対する特性マスクを探す差分攻撃である．SHA は入力として X=(X[0],X[1],…,X[15)を用いて

$$X[j] = (X[j-3] \oplus X[j-8] \oplus X[j-14] \oplus X[j-16]) \qquad (16 \le j \le 79)$$

によりメッセージを(X[0],X[1],…,X[79])に拡張する．そこで，M=(M[0], M[1],…, M[79]) とする差分マスクを考える．この差分マスクは，例えば，X=(X[0],X[1],…,X[15)に対し $\tilde{X} = X \oplus M$ が同じ値を出力するように構成される．この差分マスクを特性マスクと呼ぶ．すなわち，Chabaud と Joux による SHA に対する攻撃とはメッセージ $X$ に対する特性マスク M の探索により $\tilde{X}$ が計算できることを示したことである.

この攻撃を用いると $2^{61}$ の計算で衝突を発見できるため, バースデー攻撃より効率的な攻撃として考えられる．しかし，SHA に対する攻撃はメッセージ拡張の部分で１ビットの左巡回シフトを用いる SHA-1 には適用できないため，これは SHA-1 に対する実用的な攻撃ではない(2.4.1 節 SHA-1 の入力参照)[CJ98]．この入力拡張部分の１ビット左巡回の利用することにより SHA で線形であった入力を非線形にすることができるためその解析が難しくなる.SHA-1 に対する攻撃ではこの入力の１ビット巡回シフトを解析することが非常に重要なこととなり,この解析が行われない限り SHA-1 の圧縮関数に対する攻撃も困難である.現在,バースデー攻撃より効率のよい SHA-1 に対する実用的な攻撃はまだ報告されていない.

### ４.２.５. MDx 族ハッシュ関数に対する攻撃の比較

一般に，実用的なハッシュ関数の安全性を理論的に証明することは困難であり，通常既存の攻撃に対して耐性を持つか否かによってその安全性が評価される．したがって本報告書においても，MD5, RIPEMD-160, SHA-1 の安全性について，既存の攻撃が適用可能かという点から評価した．各ハッシュ関数に対する理想的な安全性(全数探索的攻撃に要する計算量)を表 7 で示し,実際の安全性との比較を行う.

結論から述べると，MD5，RIPEMD-160，SHA-1 は既存の実用的な攻撃に対し

て安全である（表 8，9）．表 8 は MD5，RIPEMD-160，SHA-1 の圧縮関数に対する代表的な攻撃の成否及びその攻撃のために必要な計算量を示している．表から明らかなように RIPEMD-160，SHA-1 に関しては有効な攻撃は知られていない．MD5 に関しては，初期値可変の場合の衝突および擬衝突が発見されているが，これらの発見が直ちに有効な攻撃に結びつくわけではなく，実際の安全性の観点からは問題ないと考えられている．表 9 は，上記３つのハッシュ関数およびその前バージョンである MD4，RIPEMD，SHA の衝突の探索に必要な計算量を示している．上段のバースデー攻撃は，すべてのハッシュ関数に対して適用可能な全数探索的な攻撃であり，これよりも効率的な攻撃が存在しない時そのハッシュ関数は安全（衝突困難）であるみなされる．MD4，RIPEMD，SHA に関してはバースデー攻撃よりも効率的な攻撃が存在するが，MD5，RIPEMD-160，SHA-1 に関してはそのような攻撃は報告されておらず，現在のところ安全であると考えられている．

以上述べたように，MD5，RIPEMD-160，SHA-1 に対する実用的な攻撃は知られていない．したがって，これらのハッシュ関数を実際に用いる際には，全数探索的な攻撃に対する安全性を確保しておけば十分であると考えられる．現在，ハッシュ値の長さは 160 ビット程度必要であると考えられているが，最近の計算機や攻撃法の発達を考慮して，さらに長い出力をもつハッシュ関数も提案されている[N00].

**表 7 MD5,RIPEMD-160,SHA-1 の理想的な強度**

| 攻撃の種類 | MD5 | RIPEMD-160 | SHA-1 |
|---|---|---|---|
| Preimage | $2^{128}$ | $2^{160}$ | $2^{160}$ |
| 2nd preimage | $2^{128}$ | $2^{160}$ | $2^{160}$ |
| Collision | $2^{64}$ | $2^{80}$ | $2^{80}$ |

**表 8 MD5, RIPEMD-160, SHA-1 の圧縮関数への攻撃の種類と攻撃に必要な計算量**

| 攻撃の種類 | MD5 | RIPEMD-160 | SHA-1 |
|---|---|---|---|
| Preimage | -- | -- | -- |
| 2nd preimage | -- | -- | -- |

| | | | |
|---|---|---|---|
| Collision (fixed IV) | -- | -- | -- |
| Collision (random IV) | [Dob96c] | -- | -- |
| Pseudo-collision | [BB93] | -- | -- |

( -- :攻撃が発見されていないことを示す.

[Dob96c] : MD5 の全段階（４ラウンド）に対する攻撃(3.2.2 節を参照).
Pentium PC で約 １０時間ぐらいで衝突を探索出来る.

[BB93] ： １－２ラウンドに対する攻撃(3.2.2 節を参照)
$2^{16}$ の計算量が必要である.)

**表 9 固定された初期値に対する衝突の探索に必要な計算量の比較**

| ハッシュ関数 / 攻撃 | MD4 | MD5 | RIPEMD | RIPEMD-160 | SHA | SHA-1 |
|---|---|---|---|---|---|---|
| バースデー攻撃 | $2^{64}$ | $2^{64}$ | $2^{64}$ | $2^{80}$ | $2^{80}$ | $2^{80}$ |
| 圧縮関数に対する攻撃 | $2^{22}$ | -- | $2^{31}$ | -- | $2^{61}$ | -- |

(＊ MD4 の圧縮関数に対する攻撃は Dobbertin の全段階に対する攻撃である[Dob96a].
＊ RIPEMD の圧縮関数に対する攻撃は２－３ラウンドに対する攻撃である[Dob97].
＊ SHA の圧縮関数に対する攻撃は全段階に対する攻撃である[CJ98].
＊ --はバスーデー攻撃よりいい衝突探索方法がまだ見つかっていないことを示している．)

# ５．終わりに

本報告書では実用的なハッシュ関数として現在暗号分野で用いられている MD5, RIPEMD-160, SHA-1 について,その構造の説明,それらの構成要素の比較,及び各ハッシュ関数の安全性の調査を行った.

## 参考文献


[AB96] R.Anderson and E.Biham, “Tiger : A new hash function,” Fast Software Encryption, Lecture Notes in Computer Science, Springer-Verlag, pp.89-97, 1996.

[BB91] B.den Boer and A.Bosselaers, “An attack on the last two rounds of MD4,” Advances in Cryptology-Crypto'91, Lecture Notes in Computer Science, Springer-Verlag, pp.194-203, 1991.

[BB93] B.den Boer and A.Bosselaers, "An attack on the last two rounds of MD5", Advances in Cryptology-Eurocrypt'93, Springer-Verlag, pp. 293-304 , 1993.

[BD92] F.Bauspiess and F.Damm, “Requirements for Cryptographic Hash Functions,” Computers & Security, 11, pp.427-437, 1992.

[BGR95] M.Bellare, R.Guerin, and P.Rogaway, “XOR MACs:New Methods for Message Authentication Using Finite Pseudorandom Functions,” Advances in Cryptology-Crypto'95, Lecture Notes in Computer Science,Springer-Verlag, pp.15-28,1995.

[BGV96] A.Bosselaers, R.Govaerts, and J.Vandewalle, “Fast Hashing on the Pentium”,Advances in Cryptology-Crypto'96, Lecture Notes in Computer Science, Springer-Verlag, pp.298-312，1996.

[BGV97] A.Bosselaers, R.Govaerts, and J.Vandewalle, “SHA: A Design for parallel Architectures?”,Advances in Cryptology-Eurocrypto'97, Lecture Notes in Computer Science, Springer-Verlag,pp.348-362, 1997.

[BJKS93] J.Bierbrauer, T.Johansson, G.Kabatiansukii, and B.Smeets, “On Families of Hash Functions via Geometric Codes and Concatenation,"Advances in Cryptology-Crypto'93, Lecture Notes in Computer Science, Springer-Verlag, pp.331-342, 1993.

[BKR94] M.Bellare, J.Kilian, and P.Rogaway, "The security of cipher blocks chaining," Advances in Cryptology-Crypto'94, Lecture Notes in Computer Science, Springer-Verlag, pp.341-358,1994.

[BM97] M.Bellare, and D.Micciancio, "A New Paradigm for Collision-Free Hashing: Incrementality at Reduced Cost," Advances in Cryptology-EuroCRYPTO'97, Lecture Notes in Computer Science,Springer-Verlag, pp.163-192, 1997.

[BR97] M.Bellare, and P.Rogaway, "Collision-Resistant Hashing:Towards Making UOWHFs Practical," Advances in Cryptology-Crypto'97, Lecture Notes in Computer Science, Springer-Verlag, pp.470-484,1997.

[BS91] E.Biham and A.Shamir, “Differential Cryptanalysis of DES-like Cryptosystems,” Journal of Cryptology, 4(1): pp.3-72, 1991.

[CJ98] F.Chabaud, and A.Joux, “Differential collisions in SHA-0 “, Advances in Cryptology-Crypto'98, Lecture Notes in Computer Science, Springer-Verlag , pp.56-71 , 1998.

[D89] I.B.Damgard, “A Design Principle for Hash Functions “, Advances in Cryptology-Crypto'89, Lecture Notes in Computer Science, Springer-Verlag, pp.416-427, 1989.

[DBP96] H.Dobbertin, A.Bosselaers, B.Preneel, “RIPEMD-160: A strengthened version of RIPEMD,” Fast Softeware Encryption-cambridge Workshop, LNCS vol.1039, Springer-Verlag, pp.71-82, 1996.

[DGV92] J.Daemen, R.Govaerts, and J.Vandewalle, “A Framewrok for the Design of One-Way Hash Function Including Cryptanalysis of Damgard's One-Way Function Based on a Cellular Automaton”, Advances in Cryptology-ASIACRYPT'91,Springer-Verlag, pp.82-96, 1992.

[Dob96a] H.Dobbertin, “Cryptanalysis of MD4”, Fast Software Encryption, LNCS1039, Springer-Verlag, pp.53-69, 1996.

[Dob96b] H.Dobbertin, “The status of MD5 after recent attack”, CryptoBytes, 2(2), Sep., pp.1-6, 1996.

[Dob96c] H.Dobbertin, “Cryptanalysis of MD5 compress,” Presented at the rump session of Eurocrypt’96, May 14, 1996.

[Dob97]Hans Dobbertin: “RIPEMD with Two-Round Compress Function is Not Collision-Free.” Journal of Cryptology 10(1): 51-70 ,1997.

[Dob98] H.Dobbertin, ”The First Two Rounds of MD4 Are Not One-Way,” Fast Software Encryption, Springer-Verlag, pp.284-292, LNCS1372, Springer-Verlag, 1998.

[MOV97] A.Menezes,P.C.van Oorschot, and S.A.Vanstone, “Handbook of Applied Cryptography,” CRC press, 1997.

[MZI98] M.Mihaljevic, Y.Zheng and H.Imai, “A Fast One-way Hash Function Based on Linear Cellular Automata over GF(q)”, Proceeding of Symposium on Cryptography and Information Security(SCIS)'98-6.1.D. 1998.

[MZI99] M.Mihaljevic, Y.Zheng and H.Imai, ”A Family of Fast Dedicated One-Way Hash Functions Based on Linear Cellular Automata over GF(q),” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Vol.E82-A, No.1, pp.40-47, Jan. 1999.

[N93] NIST, “Secure hash standard,” FIPS 180-1, US Department of Commerce,Washington D.C., 1993.

[N00] NIST, “New hash algorithms (SHA-256, SHA-384, and SHA-512),” http://csrc.nist.gov/cryptval/shs.html

[NKC94] S.Nandi, B.K.Kar, and P.P.Chaudhuri, “Theory and applications of cellular automata in cryptography,” IEEE Transactions on computers, vol. 43, no.12, pp.1346-1357, December, 1994.

[PGV93] B.Preneel, R.Govaerts, and Vandewalle, “Hash functions based on block ciphers: a synthetic approach,” Advances in cryptology - Crypto 93, LNCS vol.773, pp.368-378, 1994.

[PO95] B.Preneel, P.C.van Oorschot, "MDx-MAC and building fast MACs from hash functions," Advances in Cryptology-Crypto'95, Lecture Notes in Computer Science, Springer-Verlag, pp.1-4,1995.

[PQ98] J.Pieprzyk, and C.X.Qu, "Rotation-Symmetric Functions and Fast Hashing," Information Security and Privacy, Lecture Notes in Computer Science No.1438, Springer-Verlag, pp.169-180, 1998.

[R92a] R.Rivest, “The MD4 message-digest algorithm,” Request For Comments(RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April. 1992.

[R92b] R.Rivest, “The MD5 message-digest algorithm,” Request For Comments(RFC) 1321, Internet Activities Board, Internet Privacy Task Force, April 1992 (http://www.roxen.com/rfc/rfc1321.html).

[S98] D.R.Simon, “Finding collisions on a one-way street: can secure hash functions be based on general assumptions?,” Advances in Cryptology-Eurocrypt'98, Springer-Verlag, pp.334-345, 1998.

[SV94] C.P.Schnorr, and S.Vaudenay, “Black box cryptanalysis of hash networks based on multipermutaions,” Advances in Cryptology-Eurocrypt'94, Springer-Verlag, pp.47-57, 1994.

[SV98] C.P.Schnorr, and S.Vaudenay, “The black-box model for cryptographic primitives,” Journal of Cryptology 11(2), 125-140 ,1998.

[V95] S.Vaudenay, “On the need for multipermutations:cryptanalysis of MD4 and SAFER”, Fast Software Encryption-Leuven workshop, Springer-Verlag, pp.286-297, 1995.

[ZPS93] Y.Zheng, J.Pieprzyk, J.Seberry, ``HAVAL - a one-way hashing algorithm with variable length and output,” Advances in Cryptology-Auscrypt'92, LNCS Vol.718, Springer-Verlag, pp.83-104, 1993.

.