ハッシュ関数Whirlpool, RIPEMD-160の強度評価とSHA-1の改良方法の技術調査報告

株式会社 日立製作所

2006年2月27日

ハッシュ関数 Whirlpool, RIPEMD-160 の強度評価と SHA-1 の改良方法の技術調査報告

要旨: 本報告では，暗号学的ハッシュ関数として提案，実用化されている Whirlpool, RIPEMD-160 について最新の攻撃手法を考慮しながら，暗号学的ハッシュ関数としての安全性評価を行なう．さらに，別の暗号学的ハッシュ関数である SHA-1 については，これまで提案された改良方法について，技術調査報告を行なう．

Abstract: In this report, we evaluate the security of Whirlpool and RIPEMD-160 cryptographic hash functions, taking the latest attack techniques into account. In addition, we make the technical survey of modifying techniques of SHA-1 cryptographic hash function.

目 次

第1章　Whirlpoolの安全性評価

1.1　仕様と概要

1.1.1　概要

Whirlpoolは Rijmen, Barretoにより提案されたハッシュ長512ビットの暗号学的ハッシュ関数である．このハッシュ関数は NESSIE (New European Schemes for Signatures, Integrity, and Encryption) [NESSIE] で提案された [BR00]．ISO/IEC はこのハッシュ関数を標準暗号の一つとして標準化した [ISO10118-3]．

Whirlpoolは Damgård-Merkle 構成法 (DM 構成法 [M89, D89]) に基づき，圧縮関数をハッシュ関数へ変換したものである．DM 構成法で構成されたハッシュ関数の安全性はその圧縮関数に帰着されることが証明されている．さらに，圧縮関数の安全性も，その内部で用いるパラメータ付き単射な関数 (本稿では便宜上ブロック暗号と呼ぶ) の安全性に帰着することが知られている [PGV93, BRS02]．

よって，Whirlpool の安全性評価を行なう場合には，Whirlpoolが内部で用いるブロック暗号が，期待される安全性を持つかどうかを検査することが主に評価すべき点となる．

Whirlpoolはその発表から標準化策定まで 2 度の改良を行なっている．これらについては別途詳細に触れるとして，本稿の安全性評価の対象は ISO/IEC で定義されている最終版 (2006.1.5 時点) とする．

1.1.2　内部ブロック暗号

それぞれ512ビットのパラメータ (FB) とメッセージ値 (M) を入力とし，512ビットの出力 (O) を生成する．この処理の概要を図 1.1 に示す．

入力の 1024 ビットは左右それぞれ 512 ビットデータとして扱われ，各々8バイト四方の行列構造として扱う．関数内部では，以下で説明する段関数対を 10 回繰り返す処理からなる．各々の段関数対では，同一の段関数 2 度処理することで，2 つの 512 ビット入力から 2 つの 512 ビット出力を生成する．左半分 (512 ビット) 入力は固定値 ($Const^i$) をパラメータとした段関数を処理するのに対して，右半分 (512 ビット) 入力では，左半分で生成した中間値をパラメータとして段関数を処理する．

各々の段関数は図 1.2 に示すように 4 種類の関数からなる; (1)S ボックス変換, (2) バイトシフト，(3) 線形変換，(4) 排他論理和．段関数ではこれら 4 つの関数をこの順で処理する．以下に具体的な処理を記載する．

S ボックス変換では，8 ビット入出力の単射な置換表を各バイトに適用 (合計 64 回) する．バイトシフト処理では，第 i 列 ($0 \leq i \leq 7$, 最左列が 0) を下方向に i 個ずらす．線形変換では，各行を線形な変換を行なう．排他論理和では決められた値をそれぞれのバイトに排他論理和する．決められた値とは段関数のパラメータに相当し，全体構成における左のパスでは仕様で定められる固有値が用いられ，右のパスでは左のパスの決められた中間値が用いられる．

1.1.3　各バージョンの違い

NESSIE へ提案された最初のバージョンと最終的に ISO/IEC で標準化されたものとの差は 2 つの要素関数の仕様である．これらの違いを示す (NESSIE の最終的な候補アルゴリズムには，前者の S ボックスの変更のみが反映されたものである).

S-box　NESSIE に提案された最初のバージョンでは，SHA-1 による疑似乱数生成を用いた生成法による定義であった．しかし，この乱数表作成には安全性の欠陥があり線形確率が当初 $15 \cdot 2^{-6}$ として提案されていたところが，本当は $16 \cdot 2^{-6}$ であることが判明した．

```
old_sbox[256] = {
 0x68, 0xd0, 0xeb, 0x2b, 0x48, 0x9d, 0x6a, 0xe4,
 0xe3, 0xa3, 0x56, 0x81, 0x7d, 0xf1, 0x85, 0x9e,
 0x2c, 0x8e, 0x78, 0xca, 0x17, 0xa9, 0x61, 0xd5,
 0x5d, 0x0b, 0x8c, 0x3c, 0x77, 0x51, 0x22, 0x42,
 0x3f, 0x54, 0x41, 0x80, 0xcc, 0x86, 0xb3, 0x18,
 0x2e, 0x57, 0x06, 0x62, 0xf4, 0x36, 0xd1, 0x6b,
 0x1b, 0x65, 0x75, 0x10, 0xda, 0x49, 0x26, 0xf9,
 0xcb, 0x66, 0xe7, 0xba, 0xae, 0x50, 0x52, 0xab,
 0x05, 0xf0, 0x0d, 0x73, 0x3b, 0x04, 0x20, 0xfe,
 0xdd, 0xf5, 0xb4, 0x5f, 0x0a, 0xb5, 0xc0, 0xa0,
 0x71, 0xa5, 0x2d, 0x60, 0x72, 0x93, 0x39, 0x08,
 0x83, 0x21, 0x5c, 0x87, 0xb1, 0xe0, 0x00, 0xc3,
 0x12, 0x91, 0x8a, 0x02, 0x1c, 0xe6, 0x45, 0xc2,
 0xc4, 0xfd, 0xbf, 0x44, 0xa1, 0x4c, 0x33, 0xc5,
 0x84, 0x23, 0x7c, 0xb0, 0x25, 0x15, 0x35, 0x69,
 0xff, 0x94, 0x4d, 0x70, 0xa2, 0xaf, 0xcd, 0xd6,
```

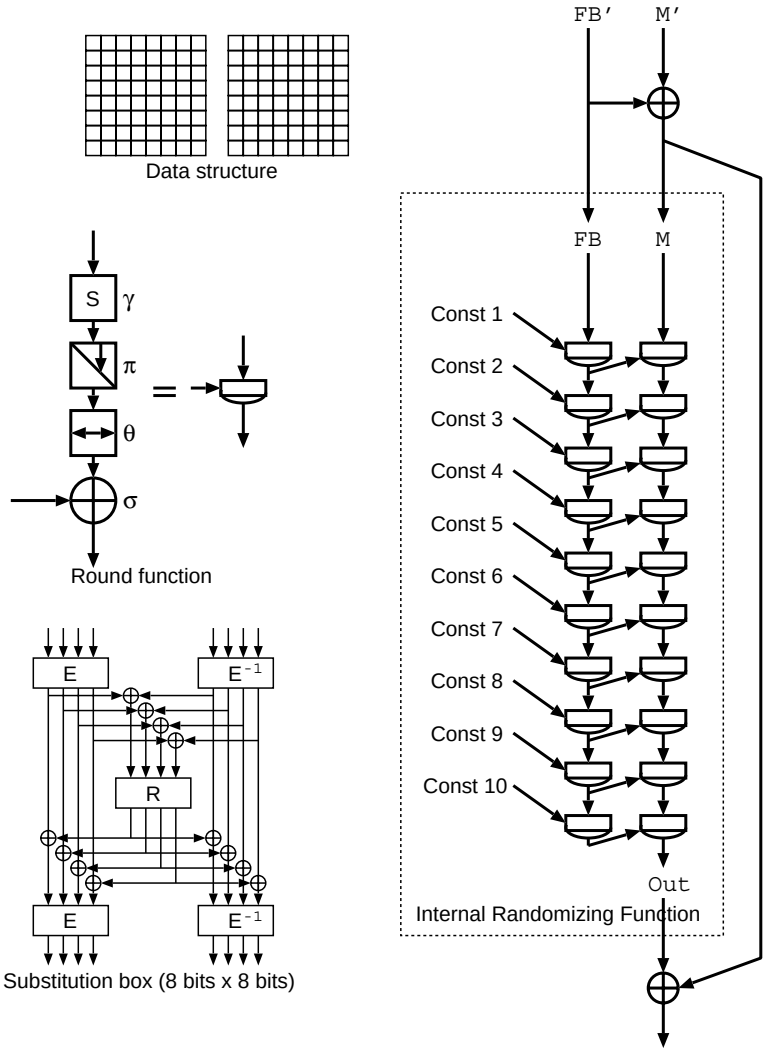

図 1.1: Whirlpool ハッシュ関数を構成する内部の処理

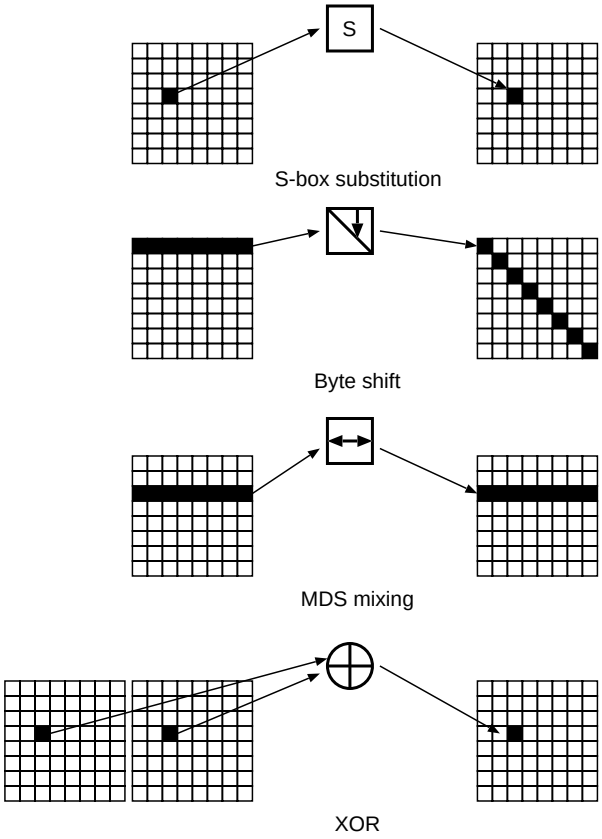

図 1.2: Whirlpool ハッシュ関数を構成する内部の処理

```
0x6c, 0xb7, 0xf8, 0x09, 0xf3, 0x67, 0xa4, 0xea,
0xec, 0xb6, 0xd4, 0xd2, 0x14, 0x1e, 0xe1, 0x24,
0x38, 0xc6, 0xdb, 0x4b, 0x7a, 0x3a, 0xde, 0x5e,
0xdf, 0x95, 0xfc, 0xaa, 0xd7, 0xce, 0x07, 0x0f,
0x3d, 0x58, 0x9a, 0x98, 0x9c, 0xf2, 0xa7, 0x11,
0x7e, 0x8b, 0x43, 0x03, 0xe2, 0xdc, 0xe5, 0xb2,
0x4e, 0xc7, 0x6d, 0xe9, 0x27, 0x40, 0xd8, 0x37,
0x92, 0x8f, 0x01, 0x1d, 0x53, 0x3e, 0x59, 0xc1,
0x4f, 0x32, 0x16, 0xfa, 0x74, 0xfb, 0x63, 0x9f,
0x34, 0x1a, 0x2a, 0x5a, 0x8d, 0xc9, 0xcf, 0xf6,
0x90, 0x28, 0x88, 0x9b, 0x31, 0x0e, 0xbd, 0x4a,
0xe8, 0x96, 0xa6, 0x0c, 0xc8, 0x79, 0xbc, 0xbe,
0xef, 0x6e, 0x46, 0x97, 0x5b, 0xed, 0x19, 0xd9,
0xac, 0x99, 0xa8, 0x29, 0x64, 0x1f, 0xad, 0x55,
```

```
 0x13, 0xbb, 0xf7, 0x6f, 0xb9, 0x47, 0x2f, 0xee,
 0xb8, 0x7b, 0x89, 0x30, 0xd3, 0x7f, 0x76, 0x82};
```

線形確率に関する上記確率をよりよいものとするために，別の手法による S ボックスの構成が行なわれた．具体的には，2 つの 4 ビット置換表 (E(とその逆変換の E^{-1}), R) を線形変換でつなぎ 8 ビットの S ボックスとするものである．E は代数的な構成により生成した変換表であり，R は疑似ランダムに生成した変換表である．これらが構成する新しい変換表を以下に示す．

```
new_sbox[256] = {
unsigned char s[256] ={
 0x18, 0x23, 0xc6, 0xe8, 0x87, 0xb8, 0x01, 0x4f,
 0x36, 0xa6, 0xd2, 0xf5, 0x79, 0x6f, 0x91, 0x52,
 0x60, 0xbc, 0x9b, 0x8e, 0xa3, 0x0c, 0x7b, 0x35,
 0x1d, 0xe0, 0xd7, 0xc2, 0x2e, 0x4b, 0xfe, 0x57,
 0x15, 0x77, 0x37, 0xe5, 0x9f, 0xf0, 0x4a, 0xda,
 0x58, 0xc9, 0x29, 0x0a, 0xb1, 0xa0, 0x6b, 0x85,
 0xbd, 0x5d, 0x10, 0xf4, 0xcb, 0x3e, 0x05, 0x67,
 0xe4, 0x27, 0x41, 0x8b, 0xa7, 0x7d, 0x95, 0xd8,
 0xfb, 0xee, 0x7c, 0x66, 0xdd, 0x17, 0x47, 0x9e,
 0xca, 0x2d, 0xbf, 0x07, 0xad, 0x5a, 0x83, 0x33,
 0x63, 0x02, 0xaa, 0x71, 0xc8, 0x19, 0x49, 0xd9,
 0xf2, 0xe3, 0x5b, 0x88, 0x9a, 0x26, 0x32, 0xb0,
 0xe9, 0x0f, 0xd5, 0x80, 0xbe, 0xcd, 0x34, 0x48,
 0xff, 0x7a, 0x90, 0x5f, 0x20, 0x68, 0x1a, 0xae,
 0xb4, 0x54, 0x93, 0x22, 0x64, 0xf1, 0x73, 0x12,
 0x40, 0x08, 0xc3, 0xec, 0xdb, 0xa1, 0x8d, 0x3d,
 0x97, 0x00, 0xcf, 0x2b, 0x76, 0x82, 0xd6, 0x1b,
 0xb5, 0xaf, 0x6a, 0x50, 0x45, 0xf3, 0x30, 0xef,
 0x3f, 0x55, 0xa2, 0xea, 0x65, 0xba, 0x2f, 0xc0,
 0xde, 0x1c, 0xfd, 0x4d, 0x92, 0x75, 0x06, 0x8a,
 0xb2, 0xe6, 0x0e, 0x1f, 0x62, 0xd4, 0xa8, 0x96,
 0xf9, 0xc5, 0x25, 0x59, 0x84, 0x72, 0x39, 0x4c,
 0x5e, 0x78, 0x38, 0x8c, 0xd1, 0xa5, 0xe2, 0x61,
 0xb3, 0x21, 0x9c, 0x1e, 0x43, 0xc7, 0xfc, 0x04,
 0x51, 0x99, 0x6d, 0x0d, 0xfa, 0xdf, 0x7e, 0x24,
 0x3b, 0xab, 0xce, 0x11, 0x8f, 0x4e, 0xb7, 0xeb,
 0x3c, 0x81, 0x94, 0xf7, 0xb9, 0x13, 0x2c, 0xd3,
 0xe7, 0x6e, 0xc4, 0x03, 0x56, 0x44, 0x7f, 0xa9,
 0x2a, 0xbb, 0xc1, 0x53, 0xdc, 0x0b, 0x9d, 0x6c,
```

```
0x31, 0x74, 0xf6, 0x46, 0xac, 0x89, 0x14, 0xe1,
0x16, 0x3a, 0x69, 0x09, 0x70, 0xb6, 0xd0, 0xed,
0xcc, 0x42, 0x98, 0xa4, 0x28, 0x5c, 0xf8, 0x86};
```

線形変換　一つは線形変換を定義する行列 C である．オリジナルでは以下のように定義されていた．下線で示す部分行列に逆行列が存在せず，この行列は安全性評価上期待される条件を満たしていないことが示された [SS03].

$$C_{\text{old}} = \begin{pmatrix} \texttt{0x01} & \texttt{0x01} & \texttt{0x03} & \texttt{0x01} & \texttt{0x05} & \texttt{0x08} & \texttt{0x09} & \texttt{0x05} \\ \texttt{0x05} & \texttt{0x01} & \texttt{0x01} & \texttt{0x03} & \texttt{0x01} & \texttt{0x05} & \texttt{0x08} & \texttt{0x09} \\ \texttt{0x09} & \underline{\texttt{0x05}} & \texttt{0x01} & \texttt{0x01} & \texttt{0x03} & \underline{\texttt{0x01}} & \texttt{0x05} & \texttt{0x08} \\ \texttt{0x08} & \texttt{0x09} & \texttt{0x05} & \texttt{0x01} & \texttt{0x01} & \texttt{0x03} & \texttt{0x01} & \texttt{0x05} \\ \texttt{0x05} & \texttt{0x08} & \texttt{0x09} & \texttt{0x05} & \texttt{0x01} & \texttt{0x01} & \texttt{0x03} & \texttt{0x01} \\ \texttt{0x01} & \underline{\texttt{0x05}} & \texttt{0x08} & \texttt{0x09} & \texttt{0x05} & \underline{\texttt{0x01}} & \texttt{0x01} & \texttt{0x03} \\ \texttt{0x03} & \texttt{0x01} & \texttt{0x05} & \texttt{0x08} & \texttt{0x09} & \texttt{0x05} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x03} & \texttt{0x01} & \texttt{0x05} & \texttt{0x08} & \texttt{0x09} & \texttt{0x05} & \texttt{0x01} \end{pmatrix}.$$

そこで，改良により以下の行列に変更された．

$$C_{\text{new}} = \begin{pmatrix} \texttt{0x01} & \texttt{0x01} & \texttt{0x04} & \texttt{0x01} & \texttt{0x08} & \texttt{0x05} & \texttt{0x02} & \texttt{0x09} \\ \texttt{0x09} & \texttt{0x01} & \texttt{0x01} & \texttt{0x04} & \texttt{0x01} & \texttt{0x08} & \texttt{0x05} & \texttt{0x02} \\ \texttt{0x02} & \texttt{0x09} & \texttt{0x01} & \texttt{0x01} & \texttt{0x04} & \texttt{0x01} & \texttt{0x08} & \texttt{0x05} \\ \texttt{0x05} & \texttt{0x02} & \texttt{0x09} & \texttt{0x01} & \texttt{0x01} & \texttt{0x04} & \texttt{0x01} & \texttt{0x08} \\ \texttt{0x08} & \texttt{0x05} & \texttt{0x02} & \texttt{0x09} & \texttt{0x01} & \texttt{0x01} & \texttt{0x04} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x08} & \texttt{0x05} & \texttt{0x02} & \texttt{0x09} & \texttt{0x01} & \texttt{0x01} & \texttt{0x04} \\ \texttt{0x04} & \texttt{0x01} & \texttt{0x08} & \texttt{0x05} & \texttt{0x02} & \texttt{0x09} & \texttt{0x01} & \texttt{0x01} \\ \texttt{0x01} & \texttt{0x04} & \texttt{0x01} & \texttt{0x08} & \texttt{0x05} & \texttt{0x02} & \texttt{0x09} & \texttt{0x01} \end{pmatrix}.$$

1.1.4　固有定数

その他実装に必要な定数を最後に示す．ブロック暗号における各段の定数 $Const^i, i = 1, \ldots, 10$ は S ボックスを使って次のように定義される．

$$\begin{aligned} Const^i_{0,c} &= s[8(i-1)+c], \quad \text{for} \quad c = 0, \ldots, 7, \\ Const^i_{r,c} &= 0, \quad \text{for} \quad r = 1, \ldots, 7, c = 0, \ldots, 7. \end{aligned}$$

なお，ハッシュ関数を構成するための，フィードバック値の初期値はバイト値 0 で埋められた 64 バイトデータである．

1.1.5 既存の安全性評価結果

Whirlpool の安全性評価を直接記載した技術文書は多くない．NESSIE への提案文書 [BR00] では，(1) 差分攻撃については個々の S ボックスの差分確率 (2^{-5}) と内部の線形変換の分岐数により最大差分特性確率が 2^{-405} を上回らないことを理由に差分攻撃に基づく攻撃法が適用できないとしている．その他文献 [GM00] や，文献 [FKSSWW01] で示された攻撃法を適用することで，内部ブロック暗号を 7 段に短縮したモデルへの攻撃が理論的には可能であると記載がある．

Whirlpool の安全性に関する技術文書として，さらに 3 点を紹介する．まず，(既に述べたが) 線形変換のエラーについての指摘は，文献 [SS03] である．S ボックスの線形確率の誤計算については NESSIE プロジェクトからのコメントによるものである [BR00]．さらに NESSIE による技術調査報告として内部ブロック暗号を 6 段に短縮した場合には，ブロック暗号として相応しくない性質があることが示された [K02]．

Whirlpool の設計と安全性評価に大きく関連するのが AES[FIPS197], Rijndael[DR98] である．AES については多くの安全性評価結果が知られている [GM00, FKSSWW01, JD04, BDK05]．

1.2 Whirlpool の安全性評価の方針

Whirlpool のハッシュ関数としての安全性評価は，前章で述べたように内部のブロック暗号に関するある種の**ランダムでない**性質を使う．よって，ブロック暗号の評価が中心となる．

これまでの評価結果や設計からの (5 年という) 年数を考慮し，本研究単体でハッシュ関数としての安全性に言及するような考察を行なうのは，必ずしも建設的な安全性評価ではなく評価として意味が少なくなる可能性がある．そこで研究の方向性として，可能な限りハッシュ関数の安全性に言及する結果を想定しながら，ブロック暗号の安全性評価として重要と考えられるものを選択し，考察を進めることにする．

1.2.1 安全性評価の着目点

まず，ハッシュ関数として重要視される安全性評価項目は差分解読法である．Whirlpool の提案文書では，差分攻撃のコストの指針として差分特性確率を使い，この上限が 2^{-204} と十分小さいことを理由に安全としている．

評価者はこの結論とその根拠ついて，再評価を行なう必要があると考える．近年のハッシュ関数に対する攻撃法 [BC04, CJ98, WLFCY05, WYY05a, WY05, WYY05b] ではいずれも，用いた差分パスから計算される差分特性確率ではなく，メッセー

ジの修正などによる劇的な差分確率の向上による効果で現実的な攻撃へと改良されているからである．上記 2^{-204} は差分特性確率に関する上限でしかなく，(無論実際に存在する差分パスとしてはこの値より小さくなることはあるが) 実際の攻撃では必ずしもこの確率が必要とは限らない．むしろ，メッセージ調整によりメッセージ 1 ビットあたり 2^{-1} の差分確率を 1 に底上げできることを考えれば 512 ビットの入力空間を考慮すれば差分特性確率 2^{-204} は不十分である．

この点を踏まえた上で，具体的に差分攻撃に基づくブロック暗号の評価を進めることにする．

1.2.2　全体構成に関する安全性評価の注意点

ブロック暗号は，2 つの入力 FB,M に対してそれぞれを単射な変換で撹拌する構成となっている．左右 512 ビットが全部ゼロ差分である状態を 0^{512}, そうでない (どこかのビットには差分がある) 状態を Δ, 差分を問わない状態を $*^{512}$ と以下で表記する．

10 段目の出力を排他論理和で結合する前の 1024 ビットの値 (I^{10}) に注目すると，FB に 0 でない差分値の入力ペアが与えられた場合 (すなわち $(\Delta, *^{512})$ タイプ)，I^{10} の左半分の差分値は常に非ゼロとなるため，$(\Delta, *^{512})$ タイプとなる (図 1.3 左)．

このことは右半分の構造についても同様のことが言える; 左半分には 0 差分が与えられた $(0^{512}, \Delta)$ タイプの場合には左半分の各段数の中間値の差分も 0^{512} となり，右半分のパラメータ部分に差分が生じないため，右半分の処理も単射な関数と見ることができ $(0^{512}, \Delta)$ タイプの出力となる (図 1.3 右)．

これらの考察がハッシュ関数においてどのような攻撃に相当するかを考える．図 1.3 右に示した $(0, \Delta)$ 入力差分は，フィードバック値に差分がないことを意味する．ハッシュ関数への攻撃として最も攻撃者に都合のよいモデルである．この攻撃モデルにさらに FB の即値として 0 値を使ってかつ，出力差分の 0 ペアを求めたとき，この攻撃はハッシュ関数の衝突となる．出力差分値が 0 でなく制御できた場合には，次のブロックの FB への差分値となる．

FB の即値として 0 値以外のものを使った場合や，FB に攻撃者にとって都合のよい差分を与えた状態で出力を 0 差分に制御した場合を考える．これはハッシュ関数への攻撃には使えず，ハッシュ関数の実質的安全性にはこのレベルでは影響がない．さらには，出力差分を 0 でなく，非ゼロ差分へ制御した場合にもそれ自身ではハッシュ関数の利用に問題は生じない．前者は疑似衝突と呼ばれる発見，後者は非疑似ランダム性の発見となり，どれも現在のハッシュ関数を構成する内部のブロック暗号の性質としてあるべきものではないと考えられている．場合によっては，将来的なハッシュ関数の利用が見送られるなどの措置が考えられる場合などがあり，それぞれ評価としては重要である．

本研究では，上記 $(0, \Delta)$ 入力差分の場合の重要性を重視し，このケースにおける

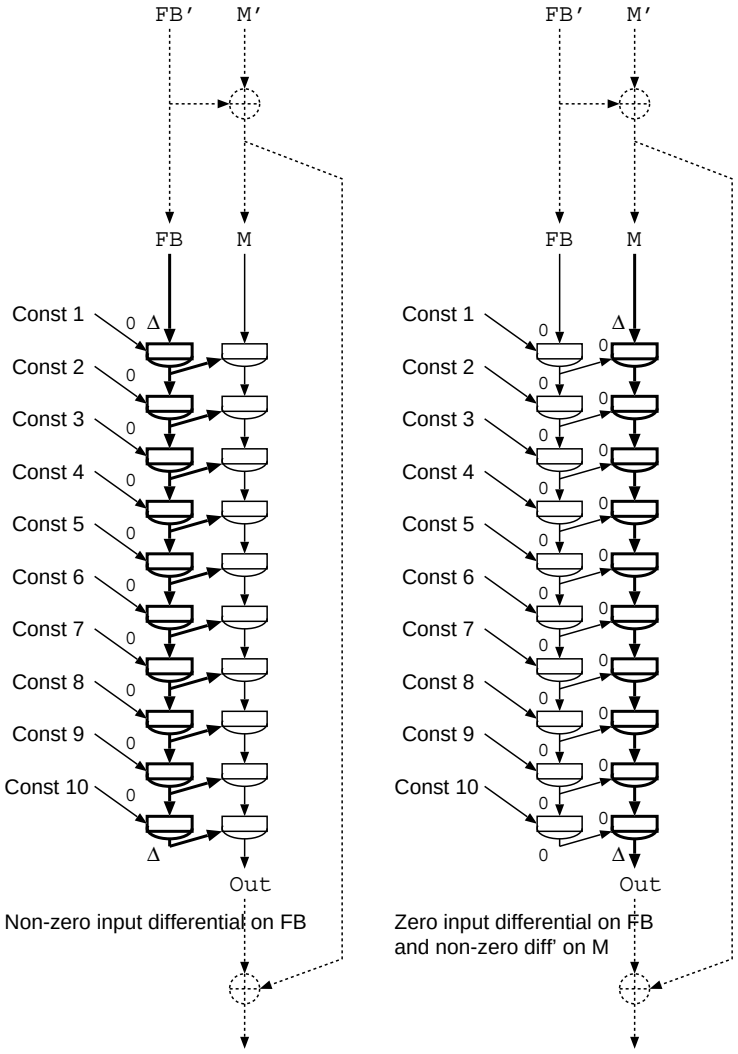

図 1.3: Whirlpool の内部ブロック暗号の大域的差分値の振舞い

差分攻撃を用いた疑似ランダム性の評価に重点を置く．そして補足的に，$(\Delta, *^{512})$ 入力差分の場合についても考察する．

1.3　バイト単位の差分特性

本稿ではデータの差分を考える場合，2 値の排他的論理和で定義される差分を考える．Whirlpool の差分特性を調べるにあたり，まずバイト単位でその特性を評価することは，全体の差分確率がどれくらいかを評価でき都合がよい．本章では，10 段 Whirlpool のバイト単位の差分特性について検討する．

差分特性には入力に必ず差分値が入る．そしてこの差分は各中間値へと伝播されてゆき，その伝播の仕方は内部の関数の仕様に応じ差分値のみから決定できる処理と，差分値からだけでは決定できない処理からなる．

バイト単位に差分特性を検討するとは，64×2 バイト毎に，差分値がある/なしのみを考えて評価することである．これにより，評価結果は厳密ではなくなるものの評価が劇的に簡単になり，これをうまく活用することで，上限値を算出するなどに使える．

1.3.1　準備

バイト単位の差分値は中間値が n バイトの場合，n ビット値で表現し，各ビットが，活性バイト (1) と不活性バイト (0) を意味する．

S ボックスは単射な関数であるので，入力バイト差分 (0,1) はそのまま出力 (0,1) となる．バイトシフトでは，差分バイトは対応するバイト処理と同じように内部状態内で移動する．

MDS は 8 バイト入力 8 バイト出力の線形変換であるが，その演算はビット単位であるので，入力バイト差分から出力バイト差分へは (実際のビット差分に依存するため) 一意には決定できない．ただし MDS としての特性により，入出力バイトの活性バイト数は常に分岐数 9 を超える．つまり入力の活性バイト数 (t_i) に対して出力で可能な活性バイト数 $(t_o \geq 9 - t_i)$ の条件がある．

パラメータ加算も MDS 同様，バイト単位の差分からは出力のバイト差分を一意に決定できない．入力 2 バイトのバイト差分が $\{(0,0),(1,0),(0,1)\}$ はそれぞれ出力バイト差分 $\{0,1,1\}$ となるが，入力バイト差分が $(1,1)$ の場合，ビット差分に応じて $\{0,1\}$ どちらも可能である．

1.3.2 対象とする撹拌関数

ハッシュ関数の安全性を議論するためには，本来圧縮関数全体である 1024 ビット入力のブロック暗号全体を見る必要がある．しかし，バイト数が多いこととその構造から，まずこの章では基本的な関数の考察として，64 バイトを撹拌する段関数列 (10 段) の，各段のパラメータに差分がない場合を考える．これにより差分特性は，Rijndael のそれに準じたものとなる．後の章節にてパラメータに差分がある場合について適宜検討する．

より厳密には，各要素関数のうち，MDS 変換とバイトシフト演算のみが，バイト単位の差分パスの評価に必要な関数となる．以下では具体的に，Whirlpool のバイト単位の差分パスを詳細に見る．

1.3.3 バイト単位の差分パス

最も基本的なバイト単位の差分パスとして，各段の入力バイト差分における活性バイト数が，1-8-64-8-1 で繰り返されるタイプがある．これは単一バイトを用いる場合には，(それ以降の段の活性バイト数の制限により) これがバイト単位として最小段数による繰り返しパターンである．この活性バイトを図示したものを図 1.4 に示す．この例における，開始/終了の単一バイトの位置は任意に選ぶことができる．このバイト差分に相当するビット差分パスが存在した場合には，4 段あたり 81(=1+8+64+8) 個の S ボックスの近似が必要となる．これを 2 回半繰り返すことにより，10 段あたりでは $171 = 2\cdot(1+8+64+8)+(1+8)$ 個が必要となる．

基本パスの変形 上記は単一バイトからの繰返し構造のあるバイト差分だった．これに準じて，複数バイト (ただし少数) が活性な段をもつバイト差分を考えることができる．これらのうち 2-7-56-16-2 の例を図 1.5 に示し，その他の場合については表 1.1 に示す．

表 1.1 では，どのパターンにおいても，活性バイト数の総和に差がないことに注意する．第 1 段と第 2 段の和が 9 であり，第 3 段と第 4 段の和が 72 である．よって，ブロック暗号の段数 (10) が繰返し段数 (4) の倍数になっていないことで若干は差がでるものの，本質的には活性バイト数では変化がない．これらはパラメータ差分がない場合であり，パラメータに差分がある場合はこの限りでない．

1.3.4 制御できる差分パスの考察

基本パスならびにその変形を考えるとバイト単位の検討による 10 段のブロック暗号では，活性バイトが $171 = 2\cdot(1+8+64+8)+(1+8)$ 個存在する．不運にもこれらすべてが S ボックスの最大差分確率 $8/256 = 2^{-5}$ で近似できたと仮定する

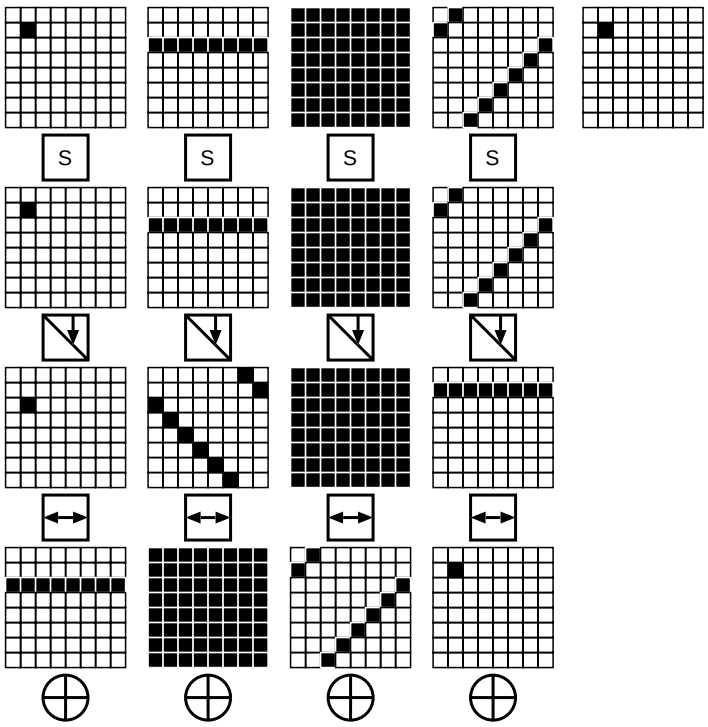

図 1.4: Whirlpool の内部ブロック暗号における活性バイト数 1-8-64-8-1 の繰返し差分構造例

表 1.1: Whirlpool の内部ブロック暗号におけるバイト単位の 4 段繰返し構造の段毎の活性バイト数

段	1	2	3	4	5	6	7	8
−1	8	16	24	32	40	48	56	64
0	1	2	3	4	5	6	7	8
1	8	7	6	5	4	3	2	1
2	64	56	48	40	32	24	16	8
3	8	16	24	32	40	48	56	64
4	1	2	3	4	5	6	7	8

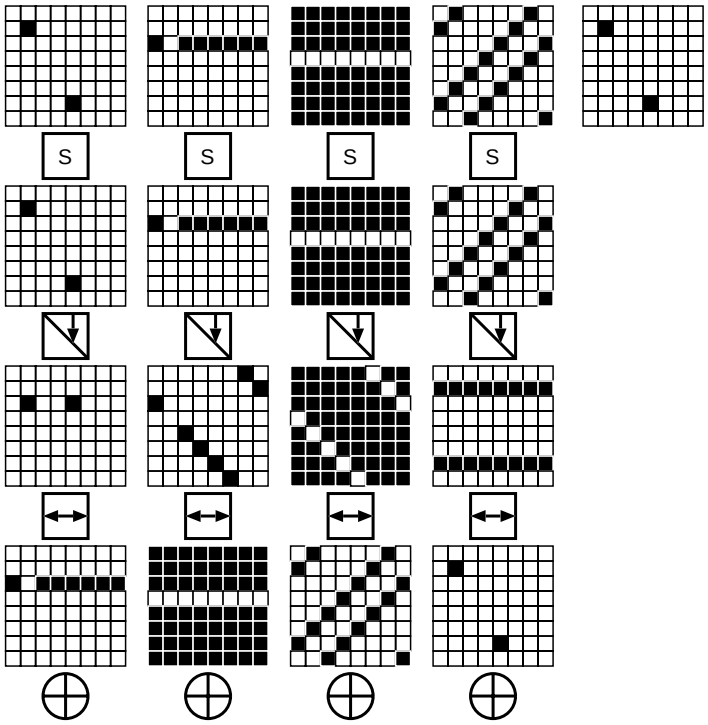

図 1.5: Whirlpool の内部ブロック暗号における活性バイト数 2-7-56-16-2 の繰返し差分構造例

と，2^{-855} の差分特性確率が期待できることとなり，圧縮関数の入力空間 1024 ビットを考えるとこの差分パスを満たす入力対の存在を否定できない．

しかし，このことからハッシュ関数 Whirlpool の衝突困難性へ言及するためにはさらなる検討が必要である; (1) ビット単位の差分パスを検討した場合差分特性確率は上記よりも小さくなると考えられる，(2) 差分パスが存在したとしても，それが圧縮関数やハッシュ関数の衝突に使えるものとは限らない．以降ではこれらについて検討する．

1.4　ビット単位の差分特性

バイト処理型の暗号プリミティブについてのビット単位の差分パスについては多くの結果が知られているが，そのほとんどが，具体的なパスの導出ではなくすべてのパスに対して言える差分 (特性) 確率の上限を評価するものである．

1024 ビットの入力空間に対して，制御可能なバイト単位の差分パスとして 2^{-855} が期待できるため，これだけでは差分攻撃 (または，これに基づく近年のハッシュ関数への攻撃法) への耐性について言及できない．

この章では有効と考えられるバイト差分パスを実現することを目的とし検討を進め，実現するビット差分のための探索とその結果について説明する．

1.4.1　モデルと目標

有効なビット単位の差分特性を導出するには，より少ない活性バイト数で実現できるバイト差分を実現させることが差分特性確率の観点から望ましい．よって，前章で示した数通りのバイト差分パスの実現を目標とする．ただし，後の理由から第 1 段，第 3 段の活性バイトの和が少ないことも安全性評価の点で重要であるので，それが最小である 1-8-64-8-1 の 4 段繰返しバイト差分の実現を目標とした．

以降の説明のために段数と S ボックス処理以下のように呼ぶ; 第 1 段から第 5 段までの差分特性を求めるが，その間の 4 つの S ボックス処理を順に，S1, S8, S64, S8' とする．

1.4.2　ビット差分導出の問題点

活性バイト数が 1-8-64-8-1 のバイト差分に対応するビット差分パスを導出することは以下の技術的理由により困難である．

まず認識するのは，活性バイト数の縮退の実現である．単一バイト入力差分は種々の変換を経たあと，第 3 段で 64 個の活性バイトが 8 個の活性バイト，かつ第 4 段でさらに 1 個にうまく縮退しなければならない．MDS や S ボックスによる差

分値の変化はこれらと関係があるものではなく単一バイト差分を変更すればほぼランダムに変化する．ランダムな 64 個の活性バイトのセットが 8 個に縮退する確率は (MDS 変換の性質から) 約 $(1/255)^{7\cdot8} \approx 2^{-448}$ となり，入力の単一差分やその後の S ボックスでの探索により生成することは不可能である．

これを回避するために個々の逆関数を用いる．具体的には，出力に相応しい単一バイト差分を特定し，これを生成する S64 出力差分を予め計算しておくことがその (縮退で膨大な探索を行なう必要性を避ける) 点においては有効である．この場合，入出力の単一バイト差分はそのバイト位置 (64 通り) と差分値 (255 通り) と，途中の S8' によるビット差分値の揺らぎ (場合の数は後述) が，それぞれ探索の範囲として動かせる部分である．

この回避方法には問題点がある．S64 では，第 1 段の単一差分 (と途中の S8 差分) で決められた入力差分値と，逆方向から同様に計算された出力差分値とがほぼランダムに生成される．ランダムな 512 ビット入出力差分を与えて，これが S64 の近似として有意である確率はとても小さくこれを探索することも不可能である．

より詳細にこれらを検討する．S1 は差分パスを形成する上での任意に差分値を動かす関数として使うほうがよいのでここでは，8-64-8 の近似部分の解析を行なう．S ボックスの差分確率の分布は，可能な入出力ペア (256^2 通り) に対して，0/256 (39655, 60.5%$\approx$ 5/8), 2/256 (20018), 4/256 (5043), 6/256 (740), 8/256 (79), 256/256 (1) である．よって単一バイトの入力差分 (出力-) を固定した場合に有意な出力差分 (入力-) 差分の場合の数はバイトあたり $256\cdot(1-39655/65536) \approx 101, 2^{6.66}$ 通り存在する．よって，単一バイト差分が S8(または S8') を通って S64 のビット差分として生成する場合の数はそれぞれ $64\cdot255\cdot2^{6.66\cdot8} \approx 2^{67.2}$ (単一バイト差分 2^{14}, S8 の差分拡散 $2^{53.2}$) となり，S64 の入出力差分の可能性は $2^{134.5}$ 通りである．この各々の差分値ペアにより，S64 が有意な差分値で近似できるのはわずかに 64 バイトすべてが有意な差分確率を持つ場合なので，$(1-39655/65536)^{64} \approx 2^{-85.785}$ となる．探索空間の成立確率から存在性は期待できるが，現実的な探索としては効率の改良が必要である．

1.4.3 繰返し表現探索アルゴリズム

効率化において重要なポイントを説明する．効率化は，現在得られた単バイト入出力を動かすことではなく，S8,S8' ボックスの差分の拡散をうまく探索に組み込むことが重要である．この差分拡散は場合の数が多く探索範囲を確保できる．

探索にはポイントが 2 点ある．1 点は探索の方針についてである．方針はいかに S64 の多くのバイトを有意な入出力差分 (good) にするか，がポイントである．ランダムな差分ならば平均 64 個中，$64\cdot(1-39655/65536) = 25.27$ 個が good であることが期待できるが，探索することによってこれを確実に増やすことを考える．

二点目のポイントは，探索の局所性を利用することである．S8(または S8') の出

力差分 8 バイトは，それぞれ独立に S64 の 1 行の差分となる．よって S64 の行単位の差分は，S8 の出力バイト 1 つを動かすことによって他のバイトに行に影響を与えることなく最適化できる．

この局所的な最適化を 2 方向で拡張する．1 つは S8(または S8') の全バイトについて適用することである．最後にこれら局所探索の結果 (S8 の出力バイトの差分) を 8 バイト一緒に合成することで，64 バイト全体の good な S ボックス数の最適化が期待できる．

二つ目の拡張は出力バイトとの連携である．出力側で (入力側の最適化の例えばあとに) 同様の最適化を行なうが，この出力差分の最適化がさらに入力差分の最適化にも影響していることに注意する．よって入力側の S8 差分，出力側の S8' 差分を交互に (局所的な最適に陥るまで) 繰り返すことで，適切な差分値を探索する．

1. 新規の第 1 段単バイト差分，第 5 段単バイト差分を選択する．
2. S8，S8' は有意な差分値により任意に選択し，S64 入出力差分値を生成し，有意な確率であるバイト数を cnt に保存する．
3. 以下のループを，S8，S8' 差分を変更しなくなるまで繰り返す．
   (a) S8 の活性バイトのうち，任意の順に以下の最適化を行なう．
   (b) ある S8 の活性バイトの出力のうち有意なものについて，S64 の入力差分を計算し，S64 の good なバイト数を記録する．
   (c) ある S8 の活性バイトの出力のうちもっとも good なバイトが多かった差分値を記録する．
   (d) S8 の 8 バイト全部の記録が終れば，これらを反映させる．反映させたとき変更したバイト数を記録しておく．
   (e) S8' についても S8 と同様に 8 つのバイトをそれぞれ独立に差分値探索を行ない，S64 の good なバイト数により評価，最適なものを反映させる．
4. S64 の全部が有意な確率で近似できていれば出力する．
5. 新規の単一バイトがなくなるまで，Step 1 へ戻って上記を繰り返す．

1.4.4　探索結果

上記探索を計算機にて実装した結果，本質的には 2 種類のパスが見つかった．また，入力差分側の単一バイトの位置は一つに固定することで本質的に同じパスを複数出力することができる．今回は入力差分側のバイト位置を 0 バイト目に固定した結果を掲載する．

計算時間は単一の PC クラスの計算機で 10 分程度であるが，コーディングにおける特別な最適化は行なっておらず，計算時間はもっと短くできる．よって計算機環境は本探索においては重要ではなく省略する．

以下に，見つかった差分パスのうちの一つを，各関数の入出力差分値として記述する．

```
Input side                Output side
10 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 46
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
          <mds>                    <mds_inv>
10 10 40 10 80 50 20 90   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   4d 83 8d 7c 7c 30 bf 05
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 00
          <s8 >                    <shift_inv>
71 fa fa 53 a5 df fd a5   00 00 00 00 7c 00 00 00
00 00 00 00 00 00 00 00   00 00 00 7c 00 00 00 00
00 00 00 00 00 00 00 00   00 00 8d 00 00 00 00 00
00 00 00 00 00 00 00 00   00 83 00 00 00 00 00 00
00 00 00 00 00 00 00 00   4d 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00   00 00 00 00 00 00 00 05
00 00 00 00 00 00 00 00   00 00 00 00 00 00 bf 00
00 00 00 00 00 00 00 00   00 00 00 00 00 30 00 00
         <shift>                   <s8'_inv>
71 00 00 00 00 00 00 00   00 00 00 00 e1 00 00 00
00 fa 00 00 00 00 00 00   00 00 00 01 00 00 00 00
00 00 fa 00 00 00 00 00   00 00 6b 00 00 00 00 00
00 00 00 53 00 00 00 00   00 f1 00 00 00 00 00 00
00 00 00 00 a5 00 00 00   90 00 00 00 00 00 00 00
00 00 00 00 00 df 00 00   00 00 00 00 00 00 00 1c
00 00 00 00 00 00 fd 00   00 00 00 00 00 00 26 00
00 00 00 00 00 00 00 a5   00 00 00 00 00 73 00 00
          <mds>                    <mds_inv>
71 71 d9 71 af a8 e2 de   e4 e4 5e e0 a3 6e 27 0b
79 fa fa cf fa 83 35 e9   c2 cb 3e 04 af 0e a4 c2
```

```
e9 79 fa fa cf fa 83 35  be 07 b1 31 18 f3 aa aa
02 a6 f1 53 53 51 53 a2  27 e3 4c c7 99 58 58 72
41 0b 57 e4 a5 a5 ae a5  7a 2f b3 76 35 35 51 b1
df b6 84 a3 69 df df 5b  b5 a8 71 cd cd 31 d2 70
d3 fd bb 2e e7 46 fd fd  f9 c6 d3 d3 d8 17 98 81
a5 ae a5 41 0b 57 e4 a5  28 48 48 84 2d d1 02 88
         (S64)                 <shift_inv>
                          e4 cb b1 c7 35 31 98 88
                          c2 07 4c 76 cd 17 02 0b
                          be e3 b3 cd d8 d1 27 c2
                          27 2f 71 d3 2d 6e a4 aa
                          7a a8 d3 84 a3 0e aa 72
                          b5 c6 48 e0 af f3 58 b1
                          f9 48 5e 04 18 58 51 70
                          28 e4 3e 31 99 35 d2 81
                                   (S64)
```

これらの差分パスで必要な各 S ボックスの近似確率を分母を 256 とした分子にて示す．なお 0 近似のものは “.” で示している．

```
     (S8)              (S8')             (S64)
2 2 2 2 2 2 2 4   . . . . 2 . . .  2 2 4 2 4 4 2 2
. . . . . . . .   . . . 4 . . . .  2 2 2 2 2 2 4 2
. . . . . . . .   . . 2 . . . . .  2 4 4 2 4 2 2 2
. . . . . . . .   . 2 . . . . . .  2 2 4 2 2 2 2 2
. . . . . . . .   2 . . . . . . .  2 2 2 2 4 2 2 2
. . . . . . . .   . . . . . . . 6  2 2 8 2 6 2 2 2
. . . . . . . .   . . . . . . 2 .  2 2 2 2 2 2 2 4
. . . . . . . .   . . . . . 2 . .  2 2 2 2 2 6 2 2
```

二つのパスはそれぞれ $(2/256)^{64}(4/256)^{12}(6/256)^{3}(8/256)^{1} = 1.68{\cdot}2^{-542}$, $(2/256)^{65}(4/256)^{12}(6/256)^{2}(8/256)$ $1.13 \cdot 2^{-543}$ の特性確率である．

1.5　ハッシュ関数の安全性

本節では，今回得られた結果に基づいてハッシュ関数 Whirlpool の安全性について考察を進める．

1.5.1 制御可能な差分パスの存在性について

本稿で得られた差分パスはその最適性は言えず，今後より高い差分確率を持つ差分パスの発見が考えられる．また，今回の差分パスについても，差分特性確率であることに注意すれば，この入出力差分ペアに対しても厳密にはより高い差分確率を持っている．これらを考慮した上で，圧縮関数に制御可能な差分パスの存在性について述べる．

劇的な差分確率の向上がなければ現状の圧縮関数には，制御可能差分が存在しない．今回の差分パスは 4 段のものであり，圧縮関数全体に適用した場合，これを 2 回半繰り返すこととなる．そのときの差分特性確率は多くとも $(1.76 \cdot 2^{-542})^2$ であり，入力空間 1024 ビットよりも大きな空間を必要とする．

また，段数を減らした場合の攻撃可能性は，7 段までは差分確率と入力空間の大小関係が逆転しない．今回のパスを使うことで 7 段の現実的な差分パスを実現することができる可能性があるがこれは今後の評価が必要である．

段数を 8 段に減らしたモデルをこの攻撃方法を適用することは不可能である．

1.5.2 圧縮関数の衝突 (疑似衝突)

上記制御可能な差分パスよりも，より困難な攻撃に圧縮関数に対する衝突 (ハッシュ関数の疑似衝突) がある．この攻撃可能性についても説明する．

圧縮関数に対して攻撃を行なうためには，圧縮関数の出力差分値を 0 にする必要がある．これには以下の 2 種類の差分のパターンが考えられる．

段関数パラメータに差分がない場合 差分パスとしては，右半分の撹拌関数のみを近似すればよく，近似する撹拌関数の数の点で攻撃しやすいが，一方でフィードフォワード 演算により出力差分を入力差分でキャンセルする必要がある (図 1.6 左)．

今回見つかった差分特性確率の高い差分パスは 4 段繰り返しでありながら，ブロック暗号は 10 段の処理であるので，入出力の差分値を一致させることができない．よって今回の差分パスはこのケースには使うことができない．

段関数パラメータに差分がある場合 段関数パラメータに差分がある場合は，入力 128 バイトのうち，左 64 バイトに差分がある場合である．

この場合は，右側の段関数列に対して各段毎に出力差分を注入するため右側の差分値はほぼ条件なく様々な値を取りうる．しかし，なるべく S ボックスの近似が少ないようなパスを考えるために局所衝突のパターンを考えて，これらを組み合わせることを考える．自明な局所衝突として，任意の段関数の入出力差分を前後で注入するものがある (図 1.7)．これは隔段で近似するため近似の効率がよい (図 1.6 右)．この場合には，左の 10 段分の段関数に加えて繰返し構造の近似を隔段で近似

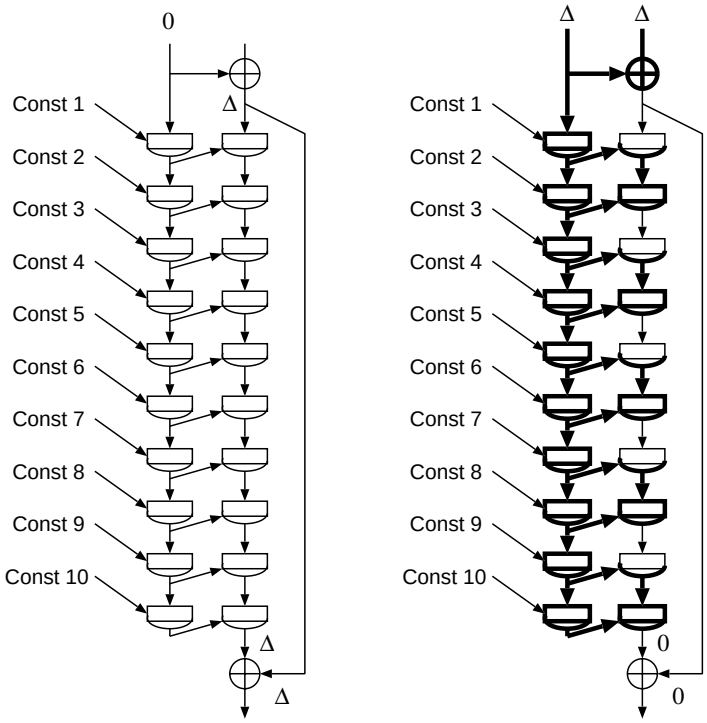

図 1.6: Whirlpool の圧縮関数における大域的な差分の伝播

する．今回示したバイト単位の差分パスの中では，隔段近似を考慮すると 1-8-64-8- の繰り返し表現が最も効率がよい．

この場合の圧縮関数に対するバイト単位の差分パスによる活性 S ボックスの数は $211 = 2 \cdot (1 + 8 + 64 + 8 + 1) + (8 + 1) + 5 \cdot 8$ となる．各々の S ボックスが最良の 2^{-5} で近似できたとしてもこの確率を満たす入力対を生成するには 1055 ビットが必要であり，1024 ビットを超えるため攻撃は適用できない．

1.5.3　ハッシュ関数の衝突

圧縮関数への攻撃よりもさらに難しいハッシュ関数に対する攻撃についても簡単に述べる．

ハッシュ関数への攻撃についてはさらに仕様で決められた初期値か，メッセージ処理中に生成される疑似ランダムな値による圧縮関数の衝突を発見する必要があり，基本的には攻撃者に許される入力空間は 512 ビットである．

今回の差分パスはそれ自身で 2^{-541} を超えるものであり，今回の結果を適用することはできない．

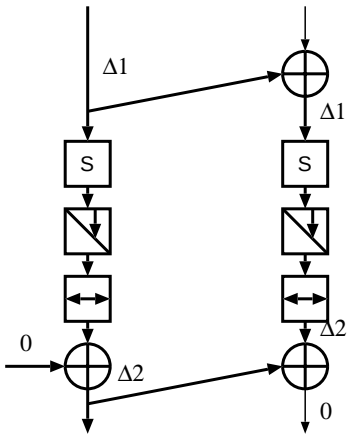

図 1.7: Whirlpool の内部ブロック暗号における自明な局所衝突の例

1.6 まとめ

本章では，ハッシュ関数 Whirlpool の安全性評価を目的とし，特に近年提案された手法の適用可能性を中心に評価を行なった．評価のポイントとして，従来の差分確率自身の現実性による評価ではなくメッセージ空間との大小関係によりこれを評価した．

この評価の第一のステップとしてバイト単位のパスの構築を行ないこの結果，ビット単位の評価結果が必要となる優良なパスを示した．続く第二のステップで実際に繰り返し表現として使えるビット単位の差分パスの探索を行ない，この差分特性確率の評価を行なった．

これらの結果，Whirlpool は圧縮関数，ハッシュ関数として十分差分確率が拡散されており，近年発見された手法を適用しても衝突データを発見することはできないと考える．

ただし，今回発見した差分パスの最適性は言えないため，今後本稿で示した結果が改良される可能性はある．

第2章　RIPEMD-160の安全性評価

2.1　仕様と概要

2.1.1　概要

RIPEMD-160はDobbertin, Bosselaers, Preneelにより提案されたハッシュ長160ビットの暗号学的ハッシュ関数である [DBP96]．ISO/IEC はこのハッシュ関数を標準暗号の一つとして標準化している [ISO10118-3]．

RIPEMD-160はDamgård-Merkle構成法 (DM構成法 [M89, D89]) に基づき，圧縮関数をハッシュ関数へ変換したものである．DM構成法で構成されたハッシュ関数の安全性はその圧縮関数に帰着されることが証明されている．圧縮関数の安全性もその内部で用いるパラメータ付き単射な関数 (本稿では便宜上ブロック暗号と呼ぶ) の安全性に帰着することが知られている [PGV93, BRS02]．

RIPEMD-160の特長として，ブロック暗号から圧縮関数の構成法が他のハッシュ関数と異なることが挙げられる．他のハッシュ関数では単一のブロック暗号に対して，フィードフォワードをかけるなどして圧縮関数を得ている [BRS02, PGV93]．RIPEMD-160ではこの方法を拡張し，異なる二つのブロック暗号を使って圧縮関数を実現している点で本質的に異なっている (図2.1)．

この構成法の違いによる安全性と効率の関係についてまとめる．効率については，従来の方式の約2倍の処理時間がかかる．ただし，二つのブロック暗号の構成を共有化できる— 特に実装ではいくつかのコスト (ソフトウェア実装ならばプログラムサイズ，ハードウェア実装ならばゲートカウント) を部品共有することで，削減することが可能である．その一方で安全性については以下の章で説明するように，一般にはより安全性が高まっていると期待されている．

RIPEMD-160の安全性評価を行なう場合には,「RIPEMD-160が内部で用いるブロック暗号が期待される安全性を持つかどうか」を検査することが主に評価すべき点となる．

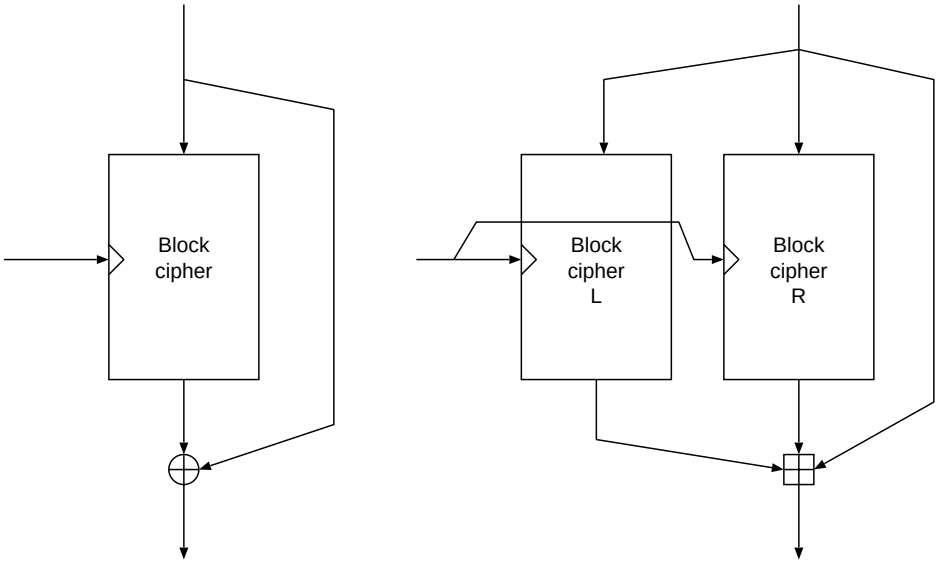

図　2.1:　圧縮関数構成法における RIPEMD-160 の特長点 (左:一般的手法, 右:RIPEMD-160 での構成法

2.1.2　圧縮関数の構成法と内部ブロック暗号 L, R

RIPEMD-160 が内部で用いる 2 個のブロック暗号 L, R とそれらから圧縮関数を構成する方法について詳細に説明する．構成を説明するにあたり，内部処理が 32 ビット単位であることから 32 ビットのデータをワードと呼ぶ．

圧縮関数，ブロック暗号ともにインターフェースは同じであり，5 ワード (160 ビット) のパラメータ (FB) と 16 ワード (512 ビット) メッセージ値 (M) を入力とし，5 ワードの出力 (O) を生成する．圧縮関数入力はそのまま 2 つのブロック暗号入力となる．圧縮関数の出力を得るために，ブロック暗号 L 出力を 2 ワード左に巡回置換したもの，ブロック暗号 R 出力を 3 ワード左に巡回置換したもの，入力を 1 ワード左に巡回置換したもの，をワード単位で算術加算する．これらのワード単位の算術加算した結果が圧縮関数の出力となる (図 2.2)．

ブロック暗号の内部処理について具体的に説明する．なお，ブロック暗号 L, R の相違点はわずかであるので適宜異なる点を明記しながら両方のブロック暗号を同時に説明する．

ブロック暗号の処理は 5 ラウンドからなり，ラウンド関数はさらに 16 ステップからなるステップ関数からなる．よって一つのブロック暗号を処理するために合計 80 ステップを処理する．

各々のステップ関数では 5 つのワード a, b, c, d, e を以下の手順で変換する．この変換に使うパラメータとして，入力であるメッセージワード M 以外に，ラウンド

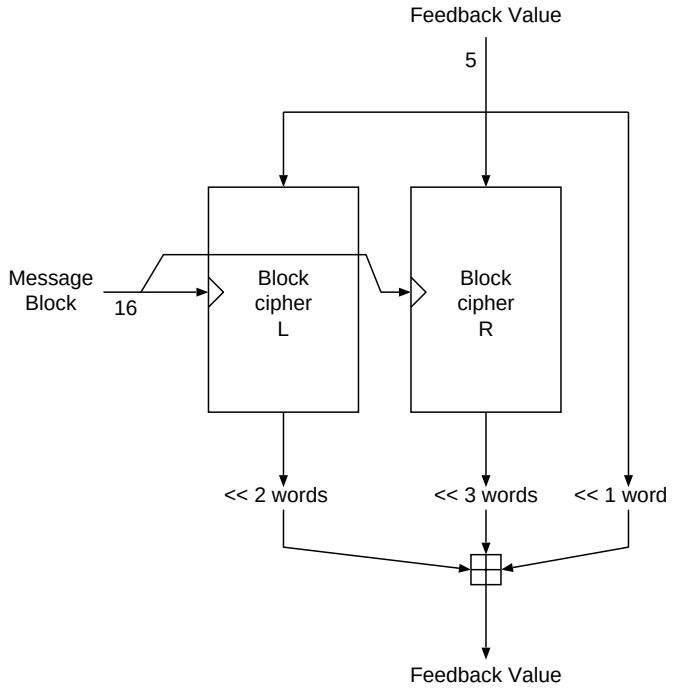

図 2.2: RIPEMD-160 の圧縮関数の構成方法の詳細

異存定数 C, ステップ異存巡回シフトオフセット t_j, ステップ異存メッセージワード インデックス a_j がある.

1. ラウンド 異存のブール関数 f による非線形処理の結果とメッセージワード M_{a_j}, 定数 C_j を a に算術加算し, この結果を t_j ビット左巡回シフトする. さらに e を加算する.
2. c を 10 ビット左巡回シフトする.
3. メッセージワード を右に 1 ワード 巡回置換する.

以上のステップ関数の動作を図 2.3 に示す. L, R のブロック暗号では C, t_j, a_j, f の定義が異なる.

2.1.3 ブール関数

RIPEMD-160 ハッシュ関数においてデータ撹拌の中心的役割の 1 つとなるのがここで説明するブール関数である. RIPEMD-160 では 3 種類のブール関数を用いながら 2 つのブロック暗号を処理する. それぞれのラウンド におけるブール関数の定義は以下のとおり.

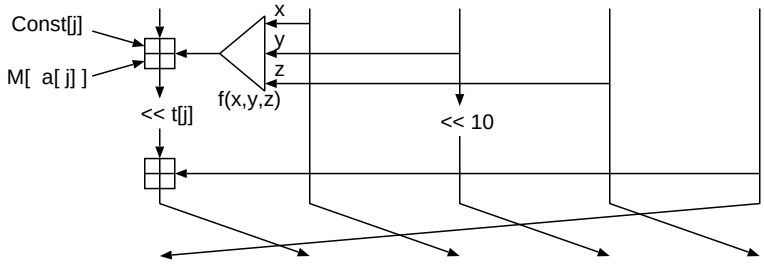

図 2.3: RIPEMD-160 の内部で用いられるステップ関数の処理

round j	L	R
$0 \le j \le 15$	$f_1(x,y,z) = x \oplus y \oplus z$	f_5
$16 \le j \le 31$	$f_2(x,y,z) = (x \cdot y) \vee (\bar{x} \cdot z)$	f_4
$32 \le j \le 47$	$f_3(x,y,z) = (x \vee \bar{y}) \oplus z$	f_3
$48 \le j \le 63$	$f_4(x,y,z) = (x \cdot z) \vee (y \cdot \bar{z})$	f_2
$64 \le j \le 79$	$f_5(x,y,z) = x \oplus (y \vee \bar{z})$	f_1

2.1.4　定数

RIPEMD-160 のハッシュ関数として定義する際に必要なフィードバック値の初期値は 5 ワード

$$\{\texttt{0x67452301}, \texttt{0xefcdab89}, \texttt{0x98badcfe}, \texttt{0x10325476}, \texttt{0xc3d2e1f0}\};$$

で定義される．ラウンド異存の定数は，L, R の順で以下のように定義される．

```
const unsigned int CL[5]=
{0x00000000, 0x5a827999, 0x6ed9eba1, 0x8f1bbcdc, 0xa953fd4e};

const unsigned int CR[5]=
{0x50a28be6, 0x5c4dd124, 0x6d703ef3, 0x7a6d76e9, 0x00000000};
```

ステップ関数における巡回シフトのオフセットは，L, R の順で以下のように定義される．

```
const int tL[80]={
11,14,15,12, 5, 8, 7, 9,  11,13,14,15, 6, 7, 9, 8,
 7, 6, 8,13,11, 9, 7,15,   7,12,15, 9,11, 7,13,12,
11,13, 6, 7,14, 9,13,15,  14, 8,13, 6, 5,12, 7, 5,
```

```
11,12,14,15,14,15, 9, 8,   9,14, 5, 6, 8, 6, 5,12,
 9,15, 5,11, 6, 8,13,12,   5,12,13,14,11, 8, 5, 6};

const int tR[80]={
 8, 9, 9,11,13,15,15, 5,   7, 7, 8,11,14,14,12, 6,
 9,13,15, 7,12, 8, 9,11,   7, 7,12, 7, 6,15,13,11,
 9, 7,15,11, 8, 6, 6,14,  12,13, 5,14,13,13, 7, 5,
15, 5, 8,11,14,14, 6,14,   6, 9,12, 9,12, 5,15, 8,
 8, 5,12, 9,12, 5,14, 6,   8,13, 6, 5,15,13,11,11};
```

メッセージワードのインデックスの定義には，以下の 2 つの置換 π, ρ, id を使う.

```
const int rho[16]=
 { 7, 4,13, 1,10, 6,15, 3,12, 0, 9, 5, 2,14,11, 8};
```

$$\pi(i) = 9i + 5 \pmod{16}, \quad i = 0, \ldots, 15.$$

$$id(i) = i \pmod{16}, \quad i = 0, \ldots, 15.$$

これらを用いてメッセージワードのインデックスは以下のように定義される．こ

round j	L	R
$0 \le j \le 15$	id	π
$16 \le j \le 31$	ρ	$\rho\pi$
$32 \le j \le 47$	ρ^2	$\rho^2\pi$
$48 \le j \le 63$	ρ^3	$\rho^3\pi$
$64 \le j \le 79$	ρ^4	$\rho^4\pi$

れらを展開したものを以下に示す.

```
int aL[80]={
 0, 1, 2, 3, 4, 5, 6, 7,   8, 9,10,11,12,13,14,15,
 7, 4,13, 1,10, 6,15, 3,  12, 0, 9, 5, 2,14,11, 8,
 3,10,14, 4, 9,15, 8, 1,   2, 7, 0, 6,13,11, 5,12,
 1, 9,11,10, 0, 8,12, 4,  13, 3, 7,15,14, 5, 6, 2,
 4, 0, 5, 9, 7,12, 2,10,  14, 1, 3, 8,11, 6,15,13};

int aR[80]={
 5,14, 7, 0, 9, 2,11, 4,  13, 6,15, 8, 1,10, 3,12,
 6,11, 3, 7, 0,13, 5,10,  14,15, 8,12, 4, 9, 1, 2,
15, 5, 1, 3, 7,14, 6, 9,  11, 8,12, 2,10, 0, 4,13,
```

```
 8, 6, 4, 1, 3,11,15, 0,   5,12, 2,13, 9, 7,10,14,
12,15,10, 4, 1, 5, 8, 7,   6, 2,13,14, 0, 3, 9,11};
```

2.1.5　既存の安全性評価結果

RIPEMD-160 の安全性評価を直接記載した技術文書は評価者の知る限り現時点では存在しない．これは構造上関連するハッシュ関数 RIPEMD-128 についても同様である．

RIPEMD-160 に関連するハッシュ関数として，RIPEMD がある．RIPEMD に対する攻撃はいくつか知られている [D97, WLFCY05]．文献 [D97] では，3 ラウンドのうち上 2 段，または下 2 段については衝突攻撃が存在することを示した．文献 [WLFCY05] では RIPEMD と MD4 の構造上の類似性に着目し，強力な MD4 に対する攻撃 (攻撃計算量 2^8) を，RIPEMD に適用することで 2^{18} の計算量で衝突を見つけることができる．

文献 [WLFCY05] の結果は新しい結果であり，この攻撃方法の RIPEMD-160 への適用可能性の検討が重要課題である．これについて報告者は直接的な攻撃の適用は不可能と考える．

文献 [WLFCY05] における RIPEMD への攻撃は，その差分パスが MD4 と共有できる点で，MD4 の攻撃を適用しやすくしている．この差分パスはいくつかの関数定義を変更することで変化するので，MD4 の攻撃を適用させないように (1) メッセージワードのインデックスを変更する，(2) ブール関数を変更する，などにより直接的な回避が可能となる．RIPEMD-160 は差分パスを考えた場合，MD4 と共有する部分が極めて限定的であり，文献 [WLFCY05] における RIPEMD や MD4 への攻撃を RIPEMD-120/160 へ直接には適用できない．

2.2　安全性評価の方針

RIPEMD-160 のハッシュ関数としての安全性評価も，内部のブロック暗号に関する評価を行なうことで，ブロック暗号に内在するランダムでない性質の有無を調べる．

ここで注意するのは，RIPEMD-160 特有の圧縮関数の構造である．この構造により圧縮関数の衝突の生成に，必ずしも単一のブロック暗号内で衝突を起こす必要がなく，最後の加算部分で差分を打ち消し合う差分パスでも衝突を生成できる．よって衝突を起こす差分パスの存在性の検証は，もっと広義に捉える必要がある; 片方のブロック暗号にて (出力差分がどんな形であれ) 制御できる差分パスの有無を検討し，これらのうち，両方のブロック暗号の出力差分が最後のワード加算で打ち消し合うものが圧縮関数の衝突を起こすと考えるほうが網羅的に安全性を検

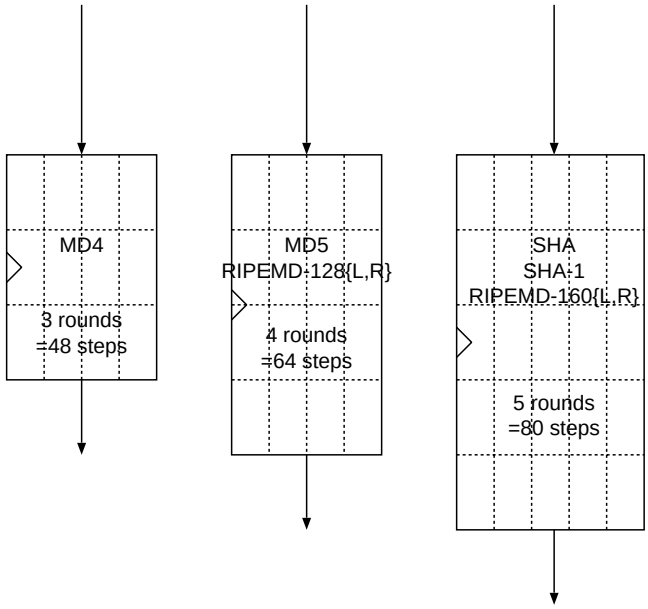

図 2.4: 各種ハッシュ関数で使う内部ブロック暗号のパラメータ比較

討したことになる．

まずは片側 (例えばブロック暗号 L) の差分パスの制御可能性について検討する—もし片方のブロック暗号における差分パスが制御できなければ，仮にもう片方がどんなに弱くとも圧縮関数として衝突を起こすことはできない．よって以降では片方のブロック暗号 L への差分パスの評価を行なう．

2.2.1 安全性評価の着目点

ブロック暗号のサイズ 評価の対象となるブロック暗号は，16 ワードの入力を使いながら 5 ワードの中間値を 80 ステップかけて更新するものである．またその関数の具体的構造も含めて，これらのパラメータは SHA[FIPS180], SHA-1[FIPS180-1] と同一である (比較図, 図 2.4)．これらのパラメータの類似性から考えられるのは，MD4, MD5, RIPEMD などのような，単一ビットによる差分パスの構成ではなく，単一のステップあたりに複数の差分ビットが含まれるような比較的複雑な差分パスとなることが考えられる．

局所衝突 RIPEMD-160 のステップ関数は SHA-1 など従来のハッシュ関数の構成に類似しており，以下ビットから構成される自明な局所衝突を考えることができる．ここで，$Z_j^{(i)}$ は，メッセージブロックの第 $j = 0, \ldots, 15$ ワードの第 $i (\mathrm{mod}\, 32)$

番目ビット (最下位を 0) を示し，これらのビットをセット (活性差分) にすることで局所衝突をなす，という意味である．

$$\{\underline{Z^{(i)}_{a_j}}, \underline{Z^{(i-t_{j-1})}_{a_{j-1}}}, Z^{(i)}_{a_{j-2}}, Z^{(i-10)}_{a_{j-3}}, Z^{(i-10)}_{a_{j-4}}, \underline{Z^{(i-t_{j-5}-10)}_{a_{j-5}}}\}$$

下線付のビットは，ブール関数の種類や即値とは関係なく必要なものである．下線無しのビットは，ブール関数の種類や即値により局所衝突に必要/不必要が決まるものである．

MD5 に対する衝突を生成した差分パスでは，第 5 ラウンドでの局所衝突と，(それ以外のラウンドにおける) 制御可能な差分パス，とに分割されてた．これによりなるべく多くのステップにゼロ差分による近似を導入することができる．

2.2.2　パス探索のアプローチ

最後にパス探索のアプローチについてまとめる．

近年知られた多くのハッシュ関数の安全性評価と同じく，まずブロック暗号を線形化したモデルを考える．これを対象にした差分パス探索を行なうことで，実際の圧縮関数のパスを構築する．

パス探索において技術的な課題が 2 点ある; 一つは圧縮関数のモデル化，もう一つは具体的な探索方法である．

圧縮関数のモデル化では，ブール関数の扱いが問題となる．RIPEMD-160 のような 5 ラウンドでメッセージ置換による圧縮関数における差分パスの振舞いがこれまで知られていないため，最初の目標として，ブール関数の近似が攻撃者にとって都合のよいものを選択できることとし，まずはステップ依存の巡回シフトとメッセージスケジュールの撹拌効果について調べることとする．

上記結果を反映させた上で，次にブール関数の具体的な近似と加算における繰り上がり (パス探索では逆関数の解析を行なうので技術的には減算における繰り下がり) を考慮したパス探索を行なう．

2.3　単純モデルとそのパス探索

2.3.1　ブロック暗号の線形モデル (1)

まず RIPEMD-160 のパス探索の最初の試みとして，RIPEMD-160 のブロック暗号を以下のように簡単にしたモデルを考える．

1. 算術加算を排他的論理和に置き換える．
2. ブール関数の差分値は (活性差分ビット位置は変更せずに) 攻撃者の都合のよい出力差分値をビット単位で選択できる．

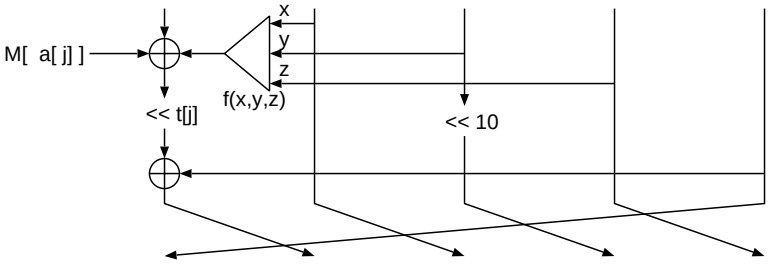

図 2.5: 差分パスを探索するための RIPEMD-160 ステップ関数の変形

このモデルに対して差分パスの探索を行なう．このモデルではラウンド毎のブール関数の性質の違いは考慮しない．差分パスの探索にあたっては，定数加算部分には差分が入らないので，探索の上ではこの処理を無視する．こうして簡単化したステップ関数を図 2.5 に示す.

2.3.2 パス探索

差分パスは (1) メッセージ差分，(2)(メッセージから展開された差分値が) ブール関数を通過時の差分特性の選択，を決定することで差分パスが一意に決定される．よってこれらの要素をパラメータに探索する．なお，ここで用いたモデルでは 32 ビットの巡回シフト演算とビット単位の論理演算のみを考えているため，これら 32 ビットの構造に対称性がある．よって，探索では探索開始のワードを任意に選びそのビット位置を固定しても一般性は失わず，探索時間を減らすことができる.

見つかった差分パスの攻撃への有効性の評価には全中間値における差分パスのハミング重みの総数 ($(160+32)\cdot 60 = 11520$ ビット分) を使うこととする．これは対象とするブロック暗号を実際に差分パスで近似した場合，最も原始的な評価では算術加算，ブール関数ともに概ね同じ確率 (0.5) での成立が期待できるからである．ただし，実際の確率は，このパスをさらに符号付差分として定義したり繰り上がり効果のないビットの算術加算差分，さらにブール関数の決定的な振舞いなどにより，より大きな確率となることが一般には期待できる.

メッセージ差分値の決定　メッセージ差分については，差分パスの最後で局所衝突を生成するパスのみを考慮した．具体的には任意の 1 ビットに差分を立て，これを逆関数方向に局所衝突を生成するものを探索した．さらに局所衝突自身もここでは 6 ビットのうち必須の 3 ビットを除いた存在可能性のある 3 ビットの活性/不活性を任意選択することとした．よって，(32 ビットの対称性により)16×8 通りのメッセージ差分が対称となる.

ブール関数特性　パス探索の経験から，一般にハミング重みを増やす選択肢は上位のステップに向けて拡散し，全体のハミング重みが増える傾向にある．よって攻撃者に好ましくないパスの傾向がある．

ブール関数の差分特性については原則差分値が通らない近似とし，通過することでその段の差分をキャンセルできるものだけを通過させることとした．

これをベースに，次のステップで差分が消えるもの (ただし通すこと自身でハミング重みが増えているので全体では増減なし) について探索をかけることとした．これにより単一の差分ビットが次のステップでどちらのレジスタに立つかを選択できる (図 2.6)．

探索の限定条件　パス探索における分岐は上記のステップ関数におけるレジスタの選択だがこれは探索においては数百もの数となり全数探索できない．そこで，本報告では数段区切りでハミング重みが最適となるパターンを選択し，これらを組み上げることで都合のよいパスの探索を行なうこととした．

2.3.3　実験結果

パス探索の結果，(メッセージ修正などで直接差分パスが必要ないステップである 1〜20 ステップを除いた)20〜80 ステップについて，中間値のハミング重みが合計 277 となるパスを生成した．

2.3.4　考察

RIPEMD-160 のブロック暗号はステップ数，レジスタ数ともに SHA/SHA-1 と同じであるのでこれと比較することができる．現在知られる SHA/SHA-1 の差分パスもここで見つかったハミング重みと概ね同じハミング重みである．SHA/SHA-1 に衝突が発見されたことを考えるとブロック暗号 L についても，この結果のみからでは衝突の可能性が期待できる．

このことから，メッセージスケジュールの違いと，ステップ依存の巡回シフトとの構造的差については，ハッシュ関数の暗号学的安全性に対して大きな差となる貢献の差も同様にないと考えられる．

しかし，この差分パスにはブール関数の近似として不可能な差分パスを利用している; 例えばステップ 70．ブール関数は $x \oplus (y \vee \bar{z})$ であり，ステップ 70 では，変数 y, z に相当する変数が 0 差分，x に相当する変数に活性差分が入っている．このとき出力差分はかならず x と同じになるが，0 差分出力で探索している．

この違いは本質的であり，具体的なブロック暗号の安全性の検証にはラウンド依存のブール関数の近似可能性を検討した探索が別途必要となる．

表 2.1: RIPEMD-160 のブロック暗号 L を線形化 (1) したモデルの差分探索結果—20〜80 ステップの合計 60 ステップについてハミング重み 277 となった．ただしこの近似には**不可能なブール関数の近似が含まれており**実際の L ではこのパスの実現は不可能.

```
st:a: t| z>          set diff-bit positions           (HW)
20:a:11|  >*10       7 828   23               21      ( 6)
21:6: 9|  >          7 828    1              *21      ( 5)
22:f: 7|  >          61718    1      *        30      ( 5)
23:3:15|  >          61718  * 1        1       30      ( 6)
24:c: 7|  >        * 61718    61718    1        8      ( 8)
25:0:12|  >*                  61718   11        8      ( 5)
26:9:15|  >  8                162728   11      * 8      ( 6)
27:5: 9|  >  8                162728  *11               ( 5)
28:2:11|  > 18               *162728   16202728         ( 8)
29:e: 7|27> 18      *         31       16202728         ( 7)
30:b:13|  >*18        2 618   31        5 62630         ( 9)
31:8:12|  >           2 618    9        5 62630*        ( 8)
32:3:11|  >          121628    9      * 5 62630  5 626  (11)
33:a:13|  >          121628  * 9                5 626  ( 7)
34:e: 6|27>         *121628   12162228           41516  (11)
35:4: 7|  >*          121     12162228           41516  ( 9)
36:9:14|  >  41516    121      0 62226         * 41516  (12)
37:f: 9|  >  41516   1131      0 62226*         141830  (12)
38:8:13|  > 142526   1131    * 0 62226  0 62226 141830  (16)
39:1:15| 0> 142526  *1131      619      0 62226  82428  (15)
40:2:14|  >*142526   25        619      0 41016  82428  (13)
41:7: 8|16> 30       25       1629      0 41016* 82428  (12)
42:0:13|  > 30        3       1629    * 0 41016 10      ( 9)
43:6: 6|  >  8        3      *1629     1317     10      ( 7)
44:d: 5|  >  8      * 3       1922     1317     20      ( 7)
45:b:12|  >* 8              1922     2327     20      ( 6)
46:5: 7|  >                   029     2327    *20      ( 5)
47:c: 5|  >                   029    *2327     23      ( 5)
48:1:11| 0>                 * 029     29       23      ( 5)
49:9:12|  >         *               29        1      ( 2)
50:b:14|  >*                        7        1      ( 2)
51:a:15|  >  1                      7      * 1      ( 3)
52:0:14|  >  1                    * 7        7      ( 3)
53:8:15|  > 11              *                 7      ( 2)
54:c: 9|  > 11      *                        17      ( 2)
55:4: 8|  >*11       11                      17      ( 3)
56:d: 9|  > 17       11                     *17      ( 3)
57:3:14|  > 17       21              *                ( 2)
58:7: 5|16> 27       21      *                        ( 3)
59:f: 6|  > 27      *21                               ( 2)
60:e: 8|27>*27                                        ( 2)
61:5: 6|  >                                  *        ( 0)
62:6: 5|  >                           *               ( 0)
63:2:12|  >                 *                         ( 0)
64:4: 9|  >         *                                 ( 0)
65:0:15|  >*                                          ( 0)
66:5: 5|  >                                  *        ( 0)
67:9:11|  >                           *               ( 0)
68:7: 6|16>                 *                         ( 1)
69:c: 8|  >         *        22                       ( 1)
70:2:13|  >*                 22                       ( 1)
71:a:12|  >                   0              *        ( 1)
72:e: 5|27>                   0      *                ( 2)
73:1:12| 0>                 * 0                       ( 2)
74:3:13|  >         *                                 ( 0)
75:8:14|  >*                                          ( 0)
76:b:11|  >                                  *        ( 0)
77:6: 8|  >                           *               ( 0)
78:f: 5|  >                 *                         ( 0)
79:d: 6|  >         *                                 ( 0)
80: :  |  >                                          ( 0)
```

表 2.2: RIPEMD-160 の選択したブール関数の即値と差分特性の一覧

x	y	z	$x \oplus (y \vee \bar{z})$	Δ	$(x \cdot y) \vee (\bar{x} \cdot z)$	Δ
0	0	0	1	0	0	0
0	0	1	0	?	0	?
0	1	0	1	?	1	?
0	1	1	1	?	1	1
1	0	0	0	1	0	?
1	0	1	1	?	1	?
1	1	0	0	?	0	?
1	1	1	0	?	1	?

2.4　より厳密なモデルと改良探索

本節では，実現可能性の高いブロック暗号の差分パスを導出するためのモデル化とパス探索を行なう．

2.4.1　ブロック暗号の線形モデル (2)

差分パス探索を行なうラウンド 2 からラウンド 5 においては本質的に 2 種類のブール関数のみを使う．探索において，探索途中のステップにて近似できないようなブール関数の差分特性をパスに組み込まないようにブール関数の差分特性を表 2.2 に簡単にまとめる．これにより 0 差分のほか，(x, y, z) の差分値組合せ 1 組について確定的に出力差分が強制される．この性質を新たに追加したブロック暗号の線形モデル (2) を評価の対象とする．

2.4.2　パス探索の改良と結果

上記のモデルに対して，従来のパス探索を行なった．上記のモデルにより，最終ラウンドにおける局所衝突に従来から 1 ビット余計なメッセージ差分が必要となる．これにより従来のパス探索方法によるとハミング重みが 921 のものまでしか見つからなかった．

パスを実際に見てみると，途中のステップで差分が消える構造が探索でみつからず，上記 921 のものでも，探索の最後にあたる第 20 ステップでは 192(=160+32) ビット中 40 ビット程度の位置が活性差分であるようなパスを構成していたことが原因である．これは，メッセージ差分増加による内部レジスタの差分値のハミン

グ重みが増加したことが直接の原因と考えることができる．

ハミング重みを軽減するために，算術加算における差分パスの拡散の性質をパス探索に導入した．具体的には，連続したビット位置の活性差分についてはこれを (減算の方向で探索しているので) 最下位に集約する差分パスを使ってパスの構築を行なった．なお，この性質を使うことにより，隣接するビットの相関を使う．よってメッセージ差分の対称性は消える．メッセージ差分は 512×8 通りを対象に探索を行なった．また，ステップ途中のブール関数の近似については従来とおり，差分を消すものは通過し，次のステップで消去するものは通過/無通過の場合分け探索をした．

新しいパス探索を行なった結果，ハミング重みが 887 に減少したがブロック暗号を L の差分の制御が期待できるほどには小さくならなかった．

2.5 まとめ

本節では，これまで解析結果の知られていなかった RIPEMD-160 について安全性評価を行なった．その方針として，内部のブロック暗号の差分パスの評価を行ない，ハッシュ関数の安全性評価への影響について必要な知見を得た．

第一の知見は，RIPEMD-160 のブロック暗号 L はその構造が SHA-1 とは若干異なるものの，撹拌の本質であるメッセージ置換とステップ依存のビットシフトからくる大域的なデータの撹拌について，SHA-1 と同程度の差分パスの存在が期待できることを示した．

第二の知見は，上記の有効なパスが具体的にブール関数の近似を検討することにより実現不可能であること．また同様の差分パスを別途探索したが限られた時間による探索では同程度のパスを構築することはできなかったことを知り得た．

以上の結果から類推すると，今後の研究の進展次第ではブロック暗号の差分制御は可能となる可能性は (少なくとも SHA-1 程度には) ある．しかし，RIPEMD-160 の衝突には，2 つのブロック暗号で同時に差分値を制御できるようなパスが少なくとも必要であり，このための技術的障壁は大きい．補足として，RIPEMD の攻撃は二個同時の差分制御に成功しているが，その圧縮関数内の二つのブロック暗号は，左右で同一の差分パスを利用できるほどの，極めて高い類似性を持っている．RIPEMD-160 にはその類似性に基づく「二つのブロック暗号の差分制御」はその構造上実現できない．

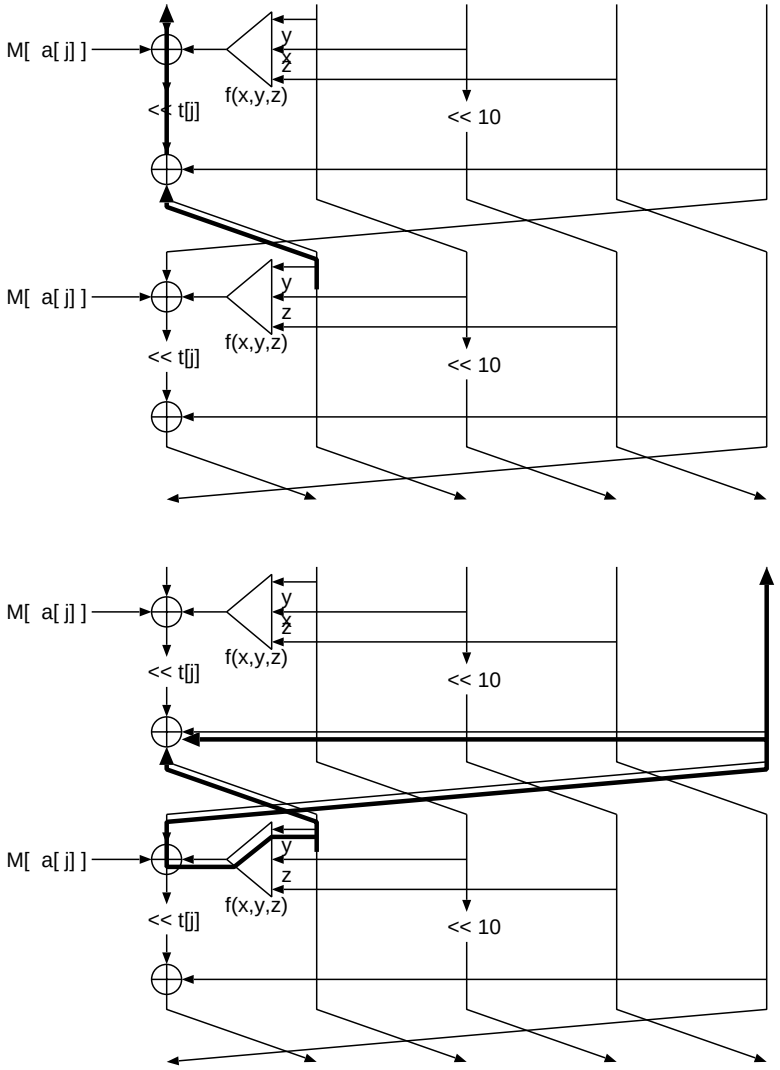

図 2.6: RIPEMD-160 の差分パス探索におけるパスの分岐

第3章　SHA-1の改良手法

3.1　はじめに

SHA-1[FIPS180-1] はデジタル署名や MAC，乱数生成など多くの安全性確保目的で広く利用されてきた暗号学的ハッシュ関数である．SHA-1 に対して，必要は計算量がかろうじて現実的なレベルの衝突発見方法が発表された [WYY05a].

この攻撃方法がもたらすハッシュ関数利用への現実的な脅威は現状限定的であるとされるが，いくつかの利用用途ではこれが直接的な脅威となる場合がある．このような用途にはより強固な暗号学的ハッシュ関数の利用が必要である．

新規の暗号学的アルゴリズムを利用することには，安全性に関する懸念が常に付きまとう—暗号プリミティブ (特にブロック暗号やストリーム暗号，ハッシュ関数などの共通鍵暗号プリミティブ) はその安全性に証明を与えることができず，その信頼は経験的なものに多かれ少なかれ依存するからである．これはハッシュ関数についても同様であり，現在安全であろうとされる SHA-224/ 256/ 384/ 512, Whirlpool への標準的利用の移行には上記の懸念がある．

このような懸念から，脆弱性が見つかりながらも安全性評価やその普及が突出している共通鍵暗号プリミティブについては脆弱性を解決するような改良案 (延命策) を考える場合がある．この例として，DES を差分解読法/線形解読法/鍵の全数探索に対して強化した T-DEA[FIPS46-3] がある．

これと同様な技術的提案を行なうことができる．この改良案の提案にはいろいろな種類のものが考えられる．この章では，これまで提案されている SHA-1 の改良案の紹介と簡単な比較考察を行なう．

3.2　改良案の分類と特徴

改良案については以下の 2 通りの分類が考えられる．

SHA-1 ハッシュ関数自身を改良するもの　メッセージスケジュールや，内部の部分関数などを変更することで発見された脆弱性などを回避する強化を行なう．改良には従来のアルゴリズムに基づいた仕様上微小な修正となることが望ましく，これにより，新たに開発するソフトウェアやハードウェアの設計時間を大きく短縮す

ることが期待できる．また修正点をうまく設計することで，新規の安全性評価を (新規にハッシュ関数を開発した場合に比較して) 減らすことが可能な場合もある．

この方法の利点として，改良手法 (案) の自由度の高さがある．ハッシュ関数自身の仕様を変更できることは，脆弱性回避のための手法にも自由度があることにつながり，この自由度は結果的に改良後の効率につながる．一般にはオリジナルの仕様とほぼ同等の処理時間や実装コストか，若干の増加での実現が期待できる．

この方法の不利点として，ハッシュ関数自身の変更を行なうため従来のライブラリや LSI を利用できないことである．これらの開発に用いたソースコードなどの資源は再利用可能であるが，すでに使用されているライブラリソフトウェアや LSI は再利用できない．

SHA-1 自身の仕様変更を伴わないもの　新たなハッシュ関数 H を定義するにあたり，脆弱性をもつ (かもしれない) ハッシュ関数 h を内部で呼び出すものである．具体的には，ハッシュ関数 H の定義のために，H のメッセージ入力に対して，適切に冗長性 (メッセージ非依存のデータ (秘密，公知なデータを含む) やメッセージ依存のデータ (メッセージの一部や，その変換データ，簡単なハッシュ値など)) を持たせる手法で，h への攻撃が，H への攻撃に直接結び付かないようにする方法である．

この方法の利点，不利点は上記「SHA-1 ハッシュ関数自身に改良するもの」の逆となる; 従来の利用資源の再利用が可能であるが，処理効率は悪くなる．

3.3　具体的手法

3.3.1　SHA-1 ハッシュ関数自身を改良するもの

メッセージスケジュール部の変更　発見された脆弱性の要点の一つとしてメッセージスケジュール部の撹拌能力の低さが挙げられる．具体的には，異なる二つの 512 ビットブロックのメッセージ対について，SHA-1 のメッセージスケジュールの仕様では (攻撃の重点的な部分となる第 20 ステップから 79 ステップまでの 60 ステップ分に相当するメッセージ伸長関数について) 最小の場合 25 ビットの差分しか発生しない．このことがデータ撹拌部分への小さい差分伝播につながり現実的な確率での差分の存在につながった．

解析のしやすい線形変換で 512 ビットを 2560 ビットへ伸長する場合，より大きな最小距離をもつ伸長方式が考えられる．このうち現状の仕様からの変更が微小でかつ上記最小距離が増加するものを探索し，それに証明が文献 [JP05] で与えら

れた.

$$W_i = \begin{cases} W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \\ \quad \oplus (W_{i-1} \oplus W_{i-2} \oplus W_{i-15})^{\lll 1}, i = 16, \ldots, 35, \\ W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16} \\ \quad \oplus (W_{i-1} \oplus W_{i-2} \oplus W_{i-15} \oplus W_{i-20})^{\lll 1}, i = 36, \ldots, 79. \end{cases}$$

3.3.2 SHA-1 自身の仕様変更を伴わないもの

ランダム化したハッシュ構成法 デジタル署名用途などに向けて，(初期値を固定した) 単一のハッシュ関数を利用せず，署名時に生成した乱数をパラメータに取るランダム化可能なハッシュ関数の提案がある．この具体的手法は SHA-1 の本質的な強化にも利用可能とされている [HK05].

具体的な方式の例として以下のものが示されている;

1. (手法 1-1) ハッシュ時，512 ビット乱数 r を生成したあと，メッセージを 512 ビット毎に分割し，各々の 512 ビットブロックを r でマスク (例えば排他的論理和) する．
2. (手法 1-2) 手法 1 にさらに加えて，メッセージ先頭に r をパディングする．
3. (手法 1-3) メッセージを 512 ビットに分割し，各々512 ビットブロックの間に r を挿入する (またはこれと手法 1 との組合せ).
4. (手法 1-4) 手法 1 をメッセージへをマスクする一方法と見てやり，その他乱数 r から (暗号学的な強弱は考慮しない) メッセージ長の擬似乱数を生成 (線形フィードバックシフトレジスタを用いたり，r への巡回シフトをブロックを超える毎に行なったものなど) し，生成系列によりメッセージをマスクするなどの手法.
5. (手法 1-5) 上記手法において 512 ビット乱数 r をより短い乱数 r_s を繰り返すなどの方法で構成する方法.

メッセージの冗長表現 SHA-1 の脆弱性を回避するために，攻撃が SHA-1 のメッセージ空間を適切に限定されるような方法は，安全性強化の方法として有効である．ここでは，文献 [SY05] で提案された手法を紹介する．

1. (手法 2-1) メッセージの適切な位置に固定のパディングを挿入する．具体的には 12 ワード毎に区切ったメッセージから，4 ワードの 0 値ワードを挿入しこれら合計 16 ワードでハッシュのブロック処理を行なう.
2. (手法 2-2) メッセージをインターリーブにより冗長化する．具体的には各メッセージワードについて，連続 2 ワードずつでハッシュ関数へ処理させる.

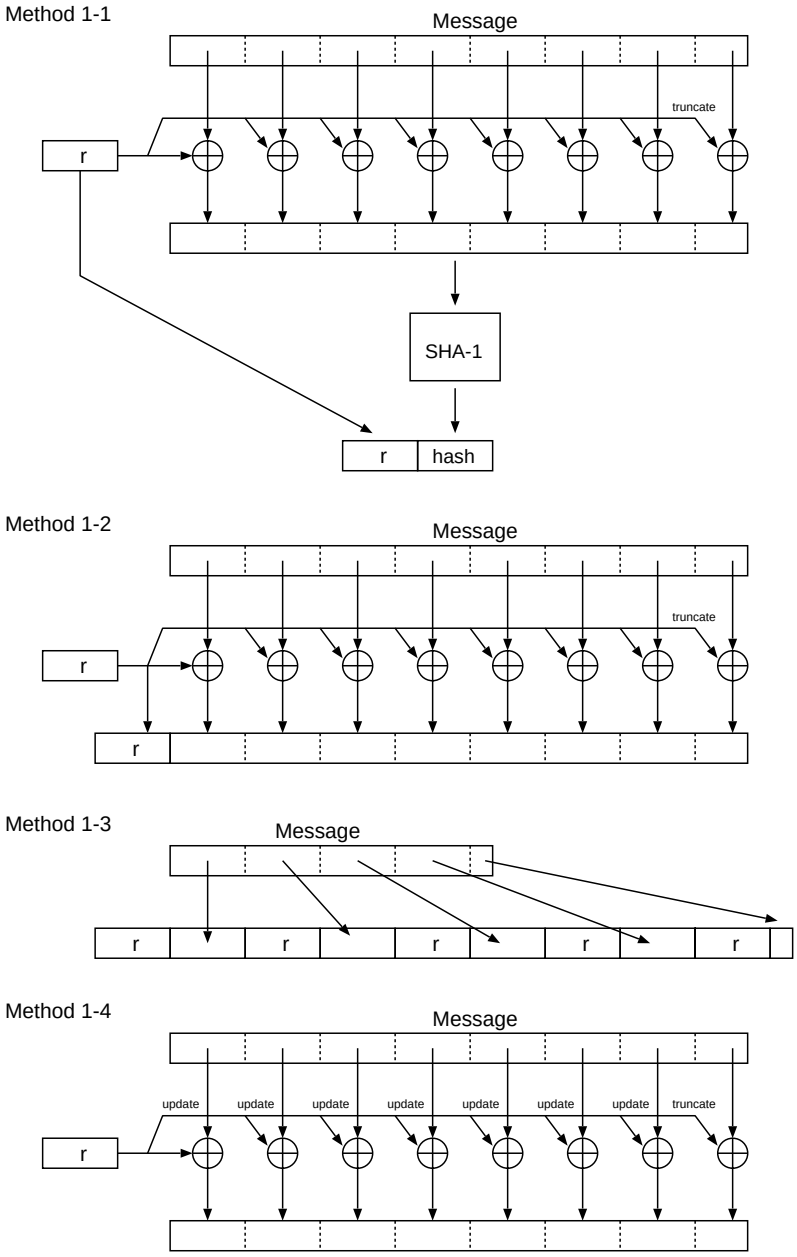

図 3.1: 乱数 r を使ったランダム化ハッシュのためのメッセージフォーマットの例

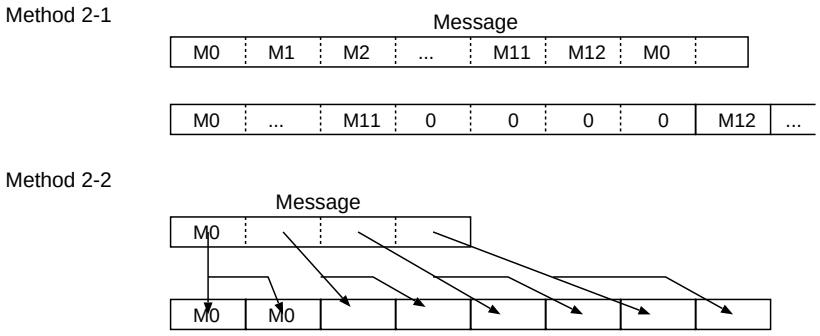

図 3.2: メッセージの冗長化による強化の例

第4章　まとめ

4.1　本報告のまとめ

本報告では，(1)Whirlpool ハッシュ関数の安全性評価，(2)RIPEMD-160 ハッシュ関数の安全性評価，(3)SHA-1 ハッシュ関数の改良案の紹介を行なった．これら安全性評価としては内部で利用するブロック暗号の安全性評価を技術的な中心の評価対象とし，この評価結果による衝突困難性についての見解を得た．この見解を得るにあたり現時点でのハッシュ関数の攻撃手法やそのためのパス探索手法などを元に解析を行なった．

Whirlpool の安全性評価と結論　Whirlpool の安全性評価では自明な局所衝突が多数存在するため，個々の局所衝突を実現するための確率について検討した．その結果，近年のハッシュ関数の解読手法に代表される差分解読法に基づく攻撃では Whirlpool の衝突を発見することはできない見通しを得た．

具体的には差分確率として有効そうなパスを導出し，そのパスを実現したメッセージ対を生成するのに必要なメッセージ空間が飽和することを確認した．

RIPEMD-160 の安全性評価と結論　RIPEMD-160 の安全性評価では具体的なパス探索を行なった．その準備的アプローチとして，(本来ブロック暗号 2 個を同時に攻撃しなければならないところを，そのうち)1 個の攻撃可能性を検討した．

この結果，本質的な構造に対する解析では，SHA-1 と同等の差分パスが存在することが確認された．しかし，ブール関数近似を含めた近似ではこれが成り立たないことを確認した．

この結果からさらにハッシュ関数全体の衝突への適用には (2 個のブロック暗号を同時に攻撃する，などの) 技術的大きな障害があり，この結果から直接 RIPEMD-160 の安全性に言及はできない．今後の安全性評価により，攻撃が高度化されることが考えられるため，今後の解析結果や技術動向に注意を払う必要がある．

SHA-1 ハッシュ関数の改良案　本報告では SHA-1 の改良案について，これまでいくつかの方法が知られている．それぞれの方式が利点，不利点があり，これらを含めて整理報告した．

参考文献

[BR00] P.S.L.M. Barreto and V. Rijmen, "The Whirlpool Hashing Function," First Open NESSIE Workshop, Leuven, Belgium, 13–14, November 2000, was revised and revised version is available at `https://www.cosic.esat.kuleuven.be/nessie/tweaks.html`.

[BC04] E. Biham and R. Chen, "New-Collisions of SHA-0," *Advances in Cryptology — CRYPTO 2004, 24th Annual International Cryptology Conference*, Lecture Notes in Computer Science vol. 3152, pp. 290–305, Springer-Verlag, 2004.

[BDK05] E. Biham, O. Dunkelman, and N. Keller, "Related-key Boomerang and Rectangle Attacks," *Advances in Cryptology — EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Lecture Notes in Computer Science vol. 3494, pp. 507–525, Springer-Verlag, 2005.

[BRS02] J. Black, P. Rogaway, and T. Shrimpton, "Block-box Analysis of the Block-cipher-based Hash-function Constructions from PGV," *Advances in Cryptology — CRYPTO'02*, Lecture Notes in Computer Science vol. 2442, pp. 320–335, Springer, 2002.

[CJ98] F. Chabaud and A. Joux, "Differential Collisions in SHA-0," *Advances in Cryptology — CRYPTO'98*, Lecture Notes in Computer Science vol. 1462, pp. 56–71, Springer, 2002.

[DR98] J. Daemen and V. Rijmen, "AES Proposal: Rijndael," *AES Round 1 Technical Evaluation CD-1, Documentation*, National Institute of Standard and Technology, 1998.

[D89] I.B. Damgård, "A Design Principle for Hash Functions," *Advances in Cryptology — CRYPTO'89*, Lecture Notes in Computer Science vol. 435, pp. 416–427, Springer-Verlag, 1989.

[FKSSWW01] N. Ferguson, J. Kelsey, S. Lucks, B. Schneier, M. Stay, D. Wagner, and D. Whiting, "Improved Cryptanalysis of Rijndael," *Fast Software Encryption 2000*, Lecture Notes in Computer Science vol. 1978, pp. 213–231, Springer-Verlag, 2001.

[DBP96] H. Dobbertin, A. Bosselaers, and B. Preneel, "RIPEMD-160, a Strengthened Version of RIPEMD," *Fast Software Encryption — FSE'96*, Lecture Notes in Computer Science vol. 1039, pp. 71–82, Springer-Verlag, 1996.

[D97] H. Dobbertin, "RIPEMD with Two-Round Compress Function is Not Collision-Free," *Journal of Cryptology* 10:1 (1997), pp. 51–70.

[GM00] H. Gilbert and M. Minier, "A Collision Attack on 7 Rounds of Rijndael," Proceedings of the 3rd AES Conference, April 13–14, 2000, New York, pp. 230–241.

[FIPS46-3] National Institute of Standards and Technology, Federal Information Processing Standards Publication 46-3, *Data Encryption Standard* (DES), 1993.

[FIPS180] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180, *Secure Hash Standard*, 1993.

[FIPS180-1] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-1, *Secure Hash Standard*, 1995.

[FIPS197] National Institute of Standards and Technology, Federal Information Processing Standards Publication 197, *Advanced Encryption Standard* (AES).

[HK05] S. Halevi and H. Krawczyk, "Strengthening Digital Signatures via Randomized Hashing," Internet-Draft, `draft-irtf-frg-rhash-00`, 2005.

[JD04] G. Jakimoski and Y. Desmedt, "Related-Key Differential Cryptanalysis of 192-bit Key AES Variants," *Selected Areas in Cryptography, 10th Annual International Workshop, SAC 2003, Ottawa, Canada, August 14–15, 2003, Revised Papers*, Lecture Notes in Computer Science vol. 3006, pp. 208-221, Springer-Verlag, 2004,

[JP05] C.S. Jutla and A.C. Patthak, "A Simple and Provable Good Code for SHA Message Expansion," *Cryptographic Hash Workshop*, October

31–November 1, 2005, Agenda – Presentations – Papers – Biographies, National Institute of Standards and Technology, 2005.

[K02] L.R. Knudsen, "Non-random Properties of Reduced-round Whirlpool," Public Reports of the NESSIE Project, Phase 2 Public Report, NES/DOC/UIB/WP5/016, available at https://www.cosic.esat.kuleuven.be/nessie/reports/.

[M89] R. Merkle, "One Way Hash Functions and DES," *Advances in Cryptology — CRYPTO'89*, Lecture Notes in Computer Science vol. 435, pp. 428–446, Springer-Verlag, 1989.

[PGV93] B. Preneel, R. Govaerts, and J. Vandewalle, "Hash Functions Based on Block Ciphers: a Synthetic Approach," *Advances in Cryptology — CRYPTO '93*, Lecture Notes in Computer Science vol. 773, Springer-Verlag, 1994.

[SS03] T. Shirai and K. Shibutani, "On the Diffusion Matrix Employed in the Whirlpool Hashing Function," Public Reports of the NESSIE Project, Phase 2 Public Report, NES/DOC/EXT/ WP5/002/1, available at https://www.cosic.esat.kuleuven.be/nessie/reports/.

[SY05] M. Szydlo and Y.L. Yin, "Collision-resistant Usage of MD5 and SHA-1 via Message Preprocessing," *Cryptographic Hash Workshop*, October 31–November 1, 2005, Agenda – Presentations – Papers – Biographies, National Institute of Standards and Technology, 2005.

[NESSIE] NESSIE, New European Schemes for Signatures, Integrity, and Encryption, https://www.cosic.esat.kuleuven.be/nessie/.

[WLFCY05] X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD," *Advances in Cryptology — EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Lecture Notes in Computer Science vol. 3494, pp. 1–18, Springer-Verlag, 2005.

[WYY05a] X. Wang, Y.L. Yin, and H. Yu, "Finding Collisions in the Full SHA-1," *Advances in Cryptology — CRYPT 2005, 25th Annual International Crypology Conference*, Lecture Notes in Computer Science vol. 3621, pp. 17–36, Springer-Verlag, 2005.

[WY05] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," *Advances in Cryptology — EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Lecture Notes in Computer Science vol. 3494, pp. 19–35, Springer-Verlag, 2005.

[WYY05b] X. Wang, H. Yu, and Y.L. Yin, "Efficient Collision Search Attacks on SHA-0," *Advances in Cryptology — CRYPT 2005, 25th Annual International Crypology Conference*, Lecture Notes in Computer Science vol. 3621, pp. 1–16, Springer-Verlag, 2005.

[Whirlpool] The Whirlpool Hash Function, `http://paginas.terra.com.br/informatica/paulobarreto/WhirlpoolPage.html`.

[ISO10118-3] ISO/IEC 10118-3:2004, Information Technology — Security Techniques — Hash-functions— Part 3: Dedicated Hash-functions, Third Edition, 2004.