ハッシュ関数の安全性に関する技術調査報告書

(株) ソニー・コンピュータエンタテインメント
盛合 志帆

平成17年2月18日

ハッシュ関数の安全性に関する技術調査報告書

2005年2月18日

要旨: 本報告書では, ハッシュ関数及びその安全性に関する動向について報告する。世の中で広く使われている代表的なハッシュ関数の各アルゴリズムについて、技術的特徴と採用・標準化動向について述べ、これまでに知られている安全性解析結果及びソフトウェア実装性能をまとめる。

Abstract: This document reports the state of hash functions and their security. For each of the widely-used hash functions, we show technical characteristics, standardization status, known results of security analysis and software performance.

目 次

第1章　はじめに

ハッシュ関数は、任意長の入力メッセージを固定長のメッセージダイジェスト (ハッシュ値) に圧縮する関数である。特に、一方向性 (one-wayness) と衝突困難性 (collision-resistance) を満たすハッシュ関数のことを暗号学的ハッシュ関数と呼ぶ。

ハッシュ関数には、鍵を入力として与えない「鍵無しハッシュ関数」(unkeyed hash function) と、鍵を入力として与える「鍵付きハッシュ関数」(keyed hash function) があるが、本報告書では鍵無しハッシュ関数のみを扱う。鍵付きハッシュ関数の代表例としてメッセージ認証コード (message authentication code) があるが、これについては、2003 年度 CRYPTREC 活動にて暗号技術監視委員会 暗号技術調査 WG で調査報告済みである [C03a]。

ハッシュ関数の代表的な構成法として、ブロック暗号を用いる構成法、ハッシュ関数専用の構成法 (dedicated design) 剰余演算 (modular arithmetic) を用いた構成法などがあるが [ISO/IEC10118-1]、本報告書では、世の中で最も広く使われている、ハッシュ関数専用の構成法に基づいて構成された「専用ハッシュ関数」を中心に技術調査を行う。

本報告書の構成　本報告書では、まず、2 章「ハッシュ関数の基礎」において、ハッシュ関数の分類や満たすべき性質、代表的な構成法、ハッシュ関数に対する攻撃法、及びハッシュ関数の用途について述べる。

次に、3 章「代表的なハッシュ関数」において、世の中で広く使われているハッシュ関数の概要、技術的特徴、仕様及び採用・標準化動向について述べる。ここで挙げるハッシュ関数はインターネット標準である RFC、国際標準規格 ISO/IEC 10118-3、米国連邦政府標準規格 FIPS180-2、そして電子政府推奨暗号リストに掲載されているアルゴリズムをカバーしている。

4 章「安全性解析」では、3 章で挙げたハッシュ関数についてこれまでに知られている解析結果をまとめる。

5 章「ソフトウェア実装性能」では、3 章で挙げたハッシュ関数について、これまでに報告されているいくつかのソフトウェア実装性能数値を示し、 6 章でまとめる。

第2章　ハッシュ関数の基礎

ハッシュ関数は、任意長の入力メッセージを固定長のメッセージダイジェスト (ハッシュ値) に圧縮する関数である。ハッシュ関数は、ハッシュ値の計算は容易であるが、もとに戻すのは困難であるという特徴をもつ。

特に、**一方向性** (one-wayness) と**衝突困難性** (collision-resistance) を満たすハッシュ関数を暗号学的ハッシュ関数 (cryptographic hash function) と呼ぶ。以降、本報告書では、暗号学的ハッシュ関数を単に「ハッシュ関数」と呼ぶ。

2.1　ハッシュ関数の分類

ハッシュ関数には、鍵を入力として与えない**鍵無しハッシュ関数** (unkeyed hash function) と鍵を入力として与える**鍵付きハッシュ関数** (keyed hash function) がある (図 2.1 参照)。

鍵無しハッシュ関数の代表的な目的は、メッセージの改竄を検知することであり、改竄検知コード (MDC: Modification/Manipulation Detection Code) 又はメッセージ完全性コード (MIC: Message Integrity Code) と呼ばれる関数が代表的である。

このうち、特に**原像計算困難性** (pre-image resistance)、**別原像計算困難性** (second pre-image resistance) (2.2 章参照) を満たす**一方向性ハッシュ関数** (one-way hash function)、**衝突困難性** (collision resistance)、**弱衝突困難性** (weak collision resistance) (2.2 章参照) を満たす**衝突困難ハッシュ関数** (collision resistant hash function) が重要である。

一方、鍵付きハッシュ関数の代表的な目的は、メッセージのソースを認証し、かつ改竄を検知することであり、**メッセージ認証コード** (MAC: Message Authentication Code) が代表的である。これについては 2003 年度 CRYPTREC 活動にて暗号技術監視委員会 暗号技術調査 WG で調査報告を行っている [C03a]。

本報告書では鍵無しハッシュ関数を対象に調査を行う。

2.2　ハッシュ関数の満たすべき性質

(暗号学的) ハッシュ関数 H は、以下の 3 つの性質を満たしていることが望まれる。

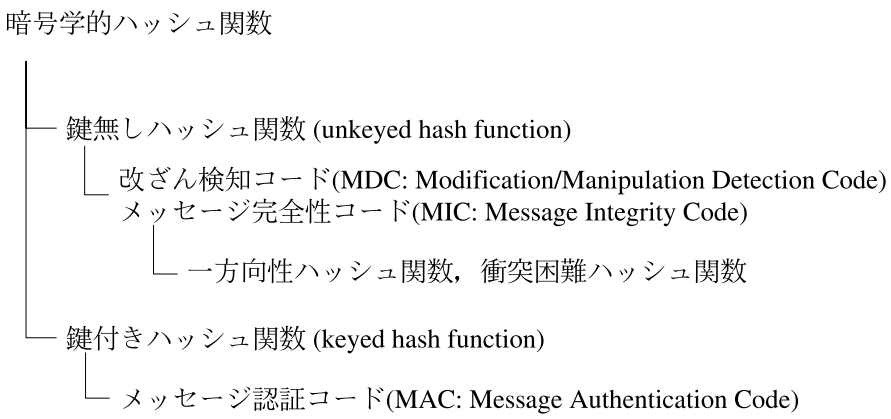

図 2.1: ハッシュ関数の分類

衝突困難性 (Collision Resistance)　ハッシュ値が一致する、すなわち $H(M) = H(M')$ を満たすような異なる 2 つのメッセージ M と M' を見つけることが計算量的に困難である。

特に、ハッシュ値 $H(M)$ と $H(M')$ の値が、わずかのビット数 (例えば 1, 2 ビット) のみしか異ならない 2 つのメッセージ M と M' を見つけることが計算量的に困難である性質のことを**近似衝突困難性** (near-collision resistance) [MOV97] と呼んでいる。

原像計算困難性 (Pre-image Resistance)　ある未知のメッセージ M に対するハッシュ値が与えられた時、ハッシュ値が一致する、すなわち $H(M) = H(M')$ を満たすようなメッセージ M' を見つけることが計算量的に困難である。この性質のことを**一方向性** (one-wayness) ともいう。

別原像計算困難性 (Second Pre-image Resistance)　ある既知のメッセージ M と M に対するハッシュ値が与えられた時、ハッシュ値が一致する、すなわち $H(M) = H(M')$ を満たすような別のメッセージ $M'(\neq M)$ を見つけることが計算量的に困難である。この性質のことを**弱衝突困難性** (weak collision resistance) ともいう。

2.3　ハッシュ関数の構成

2.3.1　繰り返し型ハッシュ関数

ハッシュ関数 $H : \{0,1\}^* \to \{0,1\}^n$ は通常、入出力が固定長の**圧縮関数** (compression function) $f : \{0,1\}^{n+m} \to \{0,1\}^n$ を繰り返し適用することで計算される。こ

のようなハッシュ関数は、**繰り返し型ハッシュ関数** (iterated hash function) または Merkle-Damgård 法 (MD 法)[M89, D89] と呼ばれる。

繰り返し型ハッシュ関数では

(1) パディング

(2) 固定長への分割

(3) 圧縮関数による繰り返し演算

(4) 出力

の順序で実行される。

(1) パディング パディング処理では、メッセージ長が m ビットの倍数になるように入力メッセージ M の末尾に下記のようにデータが付加される方法が代表的である。

$$M||1||0^k||l$$

但し、l は M のメッセージ長をバイナリ表現した時のビット数である。このように、末尾にメッセージ長を付加するパディング法を Merkle-Damgård strengthening (MD-strengthening) と呼ぶ。

(2) 固定長への分割 次にメッセージは m ビットの固定長に分割される。m ビットの固定長に分割されたメッセージを $M = (M^{(1)}||M^{(2)}||\cdots||M^{(N)}) \in (\{0,1\}^m)^*$ とする。但し $|M^{(i)}| = m$ である。

(3) 圧縮関数による繰り返し演算 m ビットの固定長に分割されたメッセージ $M^{(i)}$ が順次、圧縮関数へ入力される。

(4) 出力 最後の圧縮関数からの出力に対し、出力変換関数 g を施した結果をハッシュ値 H とする。出力変換関数 g がない (恒等変換である) 場合もある。

圧縮関数 f と出力変換関数 g を用いたハッシュ関数 $H^{fg}(H_0,\cdot)$: $(\{0,1\}^m)^* \rightarrow \{0,1\}^n$ は以下のように定義される (図 2.2 参照)。

定義 2.3.1 (繰り返し型ハッシュ関数).

$$H^{fg}(H_0,\cdot) : (\{0,1\}^m)^* \rightarrow \{0,1\}^n$$

For $i = 1$ to N
  $H_i = f(H_{i-1}, M^{(i)})$ 　　f : 圧縮関数
$H = g(H_N)$ 　　g : 出力変換関数
Return H.

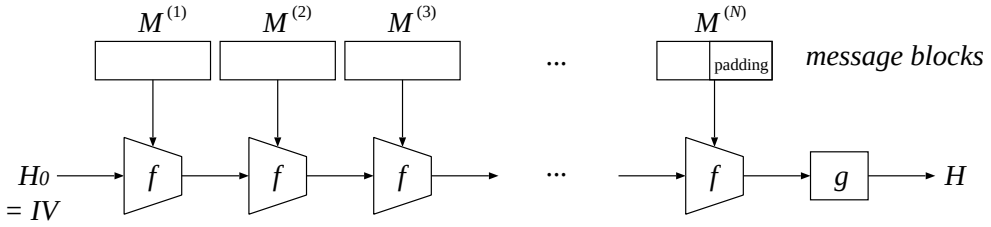

図 2.2: 繰り返し型ハッシュ関数

圧縮関数の構成法は下記のように分類することができる [ISO/IEC10118-2, ISO/IEC10118-3, ISO/IEC10118-4]。

- ブロック暗号に基づく構成法
- ハッシュ関数専用の構成法 (dedicated design)
- 剰余演算 (modular arithmetic) に基づく構成法

2.3.2　ブロック暗号に基づく構成法

ブロック暗号を圧縮関数の構成要素としてハッシュ関数を構成する場合、次のような利点が考えられる。

- 設計、評価、実装にかかるコストを削減できる
- コンパクトに実装可能

ブロック暗号に基づくハッシュ関数には、ハッシュ関数の出力長 (ハッシュ長) がブロック暗号のブロック長と同じ長さのものと、ブロック暗号のブロック長の 2 倍の長さのものがある。これらをそれぞれ、**単ブロック長ハッシュ関数** (single block length hash function)、**倍ブロック長ハッシュ関数** (double block length hash function) と呼ぶ。

単ブロック長ハッシュ関数 (Single Block Length Hash Function)　ブロック暗号 1 回当たりでハッシュされるメッセージのブロック数をレート (rate) と定義する。単ブロック長ハッシュ関数は、レート 1 の安全性が証明されたスキームが 12 種類存在することが知られており、そのうち 1 つが ISO/IEC 10118-2 に規定されている (図 2.3 参照)。単ブロック長ハッシュ関数で用いられるブロック暗号のブロック長を m とすると、この方式に対する衝突攻撃 (collision attack) の計算量は $\Omega(2^{m/2})$、(別) 原像探索攻撃 ((2nd) pre-image attack) の計算量は $\Omega(2^m)$ である。

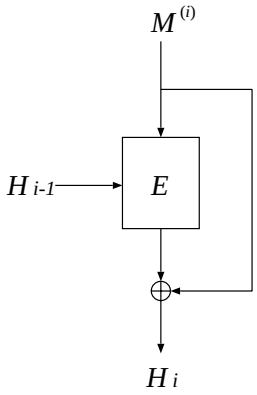

図 2.3: 単ブロック長ハッシュ関数の例

倍ブロック長ハッシュ関数 (Double Block Length Hash Function) 倍ブロック長ハッシュ関数には多くのスキームが知られており、その中で、DES ベースの 3 種類の倍ブロック長ハッシュ関数がブラックボックスモデルで衝突攻撃に対して最良の安全性をもつ (攻撃計算量が $\Omega(2^{m/2})$ である) ことが示されている [M89]。しかしながら、これらのスキームのレートは 0.276 であり、あまり効率的ではない。

衝突攻撃に対して最良の安全性をもち、よりレートの高い倍ブロック長ハッシュ関数が課題となっているが、レート 0.5 で、衝突攻撃に対して最良の安全性をもつ方式が廣瀬 [H04] らにより示されている。この方式は、鍵長がブロック長の 2 倍のブロック暗号を用いたものである。

一方、廣瀬、服部により [HH05]、レート 1 で、鍵長がブロック長の 2 倍のブロック暗号を用いた方式では、圧縮関数に計算量 $O(2^{m/2})$ の (free-start) 衝突攻撃が存在することが示された。衝突攻撃に対する最良の安全性 (計算量が $\Omega(2^{m/2})$) をもつレート 1 の倍ブロック長ハッシュ関数が存在するかはまだ未解決問題である。

2.3.3 ハッシュ関数専用構成法

ハッシュ関数専用の構成法 (dedicated design) によるハッシュ関数 (以下、専用ハッシュ関数と記す) は、表 2.1 に示すように多くのアルゴリズムが存在する。

このうち、Rivest が設計した MD4、その強度を高めた MD5 は、その後の専用ハッシュ関数の設計に大きな影響を与えた (図 2.4 参照)。

RIPEMD は 1992 年にヨーロッパの RIPE (Race Integrity Primitive Evaluation) プロジェクトの成果の一つとして提案されたハッシュ関数である [RIPE92]。RIPEMD は、MD4, 及びそれを 256 ビットハッシュ値を出力するように拡張された Extended MD4[R90] をもとに設計された。RIPEMD-160 及び RIPEMD-128 は RIPEMD の流れをくみ、さらに強度を高める目的で設計された。

HAVAL も MD4, MD5 の改良アルゴリズムで、ハッシュ長 (128, 160, 192, 224, 256 ビット) 及び処理ステップ数が可変にできる特徴をもつ。

表 2.1: さまざまな専用ハッシュ関数

名称	提案者 (機関)	提案年	ブロック長 (bit)	ハッシュ長 (bit)
MD2	Rivest	1989	512	128
MD4	Rivest	1990	512	128
MD5	Rivest	1991	512	128
RIPEMD	The RIPE Consortium	1992	512	128
RIPEMD-128	Dobertin, Bosselaers, Preneel	1996	512	128
RIPEMD-160	Dobertin, Bosselaers, Preneel	1996	512	160
HAVAL	Zheng, Pieprzyk, Seberry	1992	1024	128, 160, 192 224, 256
SHA (SHA-0)	NIST/NSA	1993	512	160
SHA-1	NIST/NSA	1995	512	160
SHA-224	NIST/NSA	2004	512	224
SHA-256	NIST/NSA	2002	512	256
SHA-384	NIST/NSA	2002	1024	384
SHA-512	NIST/NSA	2002	1024	512
Tiger	Anderson, Biham	1996	512	192
Whirlpool	Barretto, Rijmen	2000	512	512

SHA (SHA-0) 及び SHA-1 も、MD4 及び MD5 をベースに設計されたハッシュ関数である。安全性を向上するために、ハッシュ長を、当時主流であった 128 ビットから 160 ビットに拡大し、メッセージスケジュール関数が導入された。SHA-1 は SHA-0 のメッセージスケジュール関数に若干の仕様変更 (1 ビットローテーションの追加) がなされたアルゴリズムで、米国政府標準のハッシュ関数、ならびにデファクトスタンダードとして広く利用されている。さらに近年、ハッシュ長の長い SHA-224, SHA-256, SHA-384, SHA-512 も提案されている。

MD4, MD5 が 32 ビットアーキテクチャ上での高速なソフトウェア実装速度を意識して設計されたのに対し、1996 年に設計された Tiger は 64 ビットアーキテクチャを意識して設計されている。

2000 年に発表された Whirlpool は、AES の設計指針の一つである Wide Trail strategy をもとに設計されたハッシュ長 512 ビットのハッシュ関数で、メッセージ長が 2^{256} ビットまでの長い入力メッセージを扱えることが特徴である。Whirlpool は 2004 年に改訂された ISO/IEC 10118-3 に採用されている。

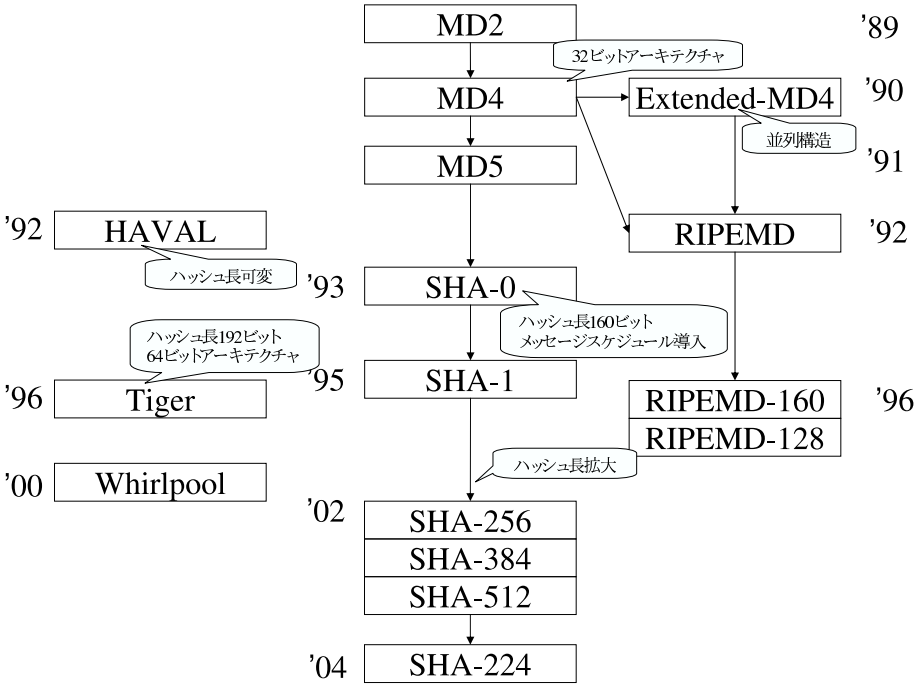

図 2.4: 専用ハッシュ関数の系譜

2.3.4 剰余演算に基づく構成法

代数構造を利用したハッシュ関数には、これまでに剰余演算 (modular arithmetic) を用いるものやナップザック問題に帰着されるものが提案されている。

加法型ナップザック問題 (additive knapsacks) に基づく構成法は、ハッシュ関数として実用的なパラメータではナップザック問題が解けてしまうのではという懸念があり、乗法型ナップザック問題 (multiplicative knapsacks) に基づく構成法は、衝突攻撃に対する安全性の証明が可能な方式もあるが、その代数構造を利用した攻撃があるのではという懸念があるなどの理由でほとんど使われていない。

剰余演算に基づくハッシュ関数としては、これまでにいろいろな方式が提案されたが、破られた方式も存在する。証明可能安全性をもつ方式もいくつか存在するが、速度が遅く、あまり実用的ではない。ここでは、ISO/IEC 10118-4 [ISO/IEC10118-4] に採用されている MASH-1, MASH-2 (Modular Arithmetic Secure Hash) について示す。

MASH-1

$$H_i = ((((m_i \oplus h_{i-1}) \vee A)^2 \pmod N) \dashv n) \oplus H_{i-1}$$

但し、$A = \texttt{F00...00}$ の定数、N は $N = pq$ (p, q はランダムに選ばれた秘密の素数) なる法である。また、$\dashv n$ は $|N|$ ビットデータの右 n ビット部分のみを左寄せする

(truncate する) 演算を示す。

MASH-2

$$H_i = ((((m_i \oplus h_{i-1}) \vee A)^{2^8+1} \pmod N) \dashv n) \oplus H_{i-1}$$

MASH-2 は MASH-1 の冪乗剰余演算の指数のみが異なる (2 が 2^8+1 となっている) アルゴリズムである。n ビットの法に対し、原像探索攻撃の計算量は $\Omega(2^{n/2})$、衝突攻撃の計算量は $\Omega(2^{n/4})$ であることが知られている。

2.4 ハッシュ関数に対する攻撃法

ハッシュ関数に対する汎用の攻撃 (generic attacks) としては、2.2 章で挙げた各項目に対応する攻撃が存在する。ハッシュ関数 $H : \{0,1\}^* \to \{0,1\}^n$ のそれぞれの攻撃に対する強度には上限が存在し、その攻撃計算量の上限はハッシュ長 n にのみ依存する。

衝突攻撃 (Collision Attack) ハッシュ値が一致する、すなわち $H(M) = H(M')$ を満たすような異なる 2 つのメッセージ M と M' を探索する攻撃。攻撃計算量は n ビットデータに対する Birthday Attack の計算量 $\Omega(2^{n/2})$ となる。

また、2 つのハッシュ値 $H(M)$ 及び $H(M')$ が、わずかのビット位置を除いて一致するような衝突を、**近似衝突 (near-collision)** という。また、繰り返し型ハッシュ関数において、初期ベクトル IV を任意に選べる条件下でハッシュ値が一致する、すなわち

$$H(IV, M) = H(IV', M')$$

となるような衝突を、**擬似衝突 (pseudo-collision)** という。

原像探索攻撃 (Pre-image Attack) ある未知のメッセージ M に対するハッシュ値が与えられた時、ハッシュ値が一致する、すなわち $H(M) = H(M')$ を満たすようなメッセージ M' を探索する攻撃。攻撃計算量は n ビットデータに対する全数探索の計算量 $\Omega(2^n)$ となる。

別原像探索攻撃 (Second Pre-image Attack) ある既知のメッセージ M と M に対するハッシュ値が与えられた時、ハッシュ値が一致する、すなわち $H(M) = H(M')$ を満たすような別のメッセージ $M'(\neq M)$ を探索する攻撃。攻撃計算量は n ビットデータに対する全数探索の計算量 $\Omega(2^n)$ となる。

2.5 ハッシュ関数の利用

ハッシュ関数は、メッセージのダイジェストを計算するという目的の他に、暗号スキームまたは暗号アルゴリズムの構成要素として利用されることが多い。ハッシュ関数の用途としては下記のようなものが代表的である。

- デジタル署名 (ほぼ全てのアルゴリズム)
- 公開鍵暗号 (例: RSA-OAEP, RSAES-PKCS1-v1_5 などのスキーム)
- 擬似乱数生成器 (例: FIPS 186-2)
- メッセージ認証コード (例: HMAC)
- ブロック暗号 (例: SHACAL-2, BEAR, LION)
- ストリーム暗号 (例: SEAL)

ハッシュ関数の入力メッセージに変更を加えると、極めて高い確率で異なるメッセージダイジェスト（ハッシュ値）が出力される。ハッシュ関数を用いたメッセージ認証コードやデジタル署名ではこれを利用して認証（メッセージの改竄検出）を行うことができる。

また、電子政府推奨暗号リスト中で利用されるハッシュ関数を表 2.2 に示す。

表 2.2: 電子政府推奨暗号リスト中で利用されるハッシュ関数

技術分類		名称	ハッシュ関数
公開鍵暗号	デジタル署名	DSA	SHA-1
		ECDSA	指定なし
		RSASSA-PKCS1-v1_5	MD5 注)
		RSAPSS	指定なし
	守秘	RSA-OAEP	指定なし
	鍵共有	PSEC-KEM	SHA-1
擬似乱数生成系		PRNG based on SHA-1 in ANSI X9.42-2001 Annex C.1	SHA-1
		PRNG based on SHA-1 for general purpose in FIPS 186-2 (+change notice 1) Appendix 3.1	SHA-1
		PRNG based on SHA-1 for general purpose in FIPS 186-2 (+change notice 1) revised Appendix 3.1	SHA-1

注) RSASSA-PKCS1-v1_5(RSA 署名, 電子署名法に係る指針に記載された方式) は、PKCS#1 v1.5 で規定され、PKCS#1 v2.1 にも引き継がれている。

1. PKCS#1 v1.5 には、MD5 についての記述 (OID) はあるが、SHA-1 についての記述 (OID) はない。
2. PKCS#1 v2.0 には、EMSA-PKCS1-v1_5 エンコーディング手法として、新たに SHA-1 が利用できるように OID が記述されている。
3. PKCS#1 v2.1 には、EMSA-PKCS1-v1_5 エンコーディング手法として、新たに SHA-256, 384, 512, 224 が利用できるように OID が記述されている。

第3章　代表的なハッシュ関数

本章では、代表的なハッシュ関数として MD4, MD5, RIPEMD, RIPEMD-128, RIPEMD-160, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512 について、その概要と技術仕様について示す。

代表的なハッシュ関数の特徴　表 3.1 に代表的なハッシュ関数の特徴をまとめる。

表 3.1: 代表的なハッシュ関数の特徴

名称	ハッシュ長 (bit)	メッセージ長 (bit)	ブロック長 (bit)	エンディアン	標準
MD4	128	上限なし	512	little	RFC 1320
MD5	128	上限なし	512	little	RFC 1321
RIPEMD	128	上限なし	512	little	
RIPEMD-128	128	$< 2^{64}$	512	little	ISO/IEC 10118-3
RIPEMD-160	160	$< 2^{64}$	512	little	電子政府推奨暗号 注) ISO/IEC 10118-3
SHA-1	160	$< 2^{64}$	512	big	電子政府推奨暗号 注) FIPS 180-2 ISO/IEC 10118-3
SHA-224	224	$< 2^{64}$	512	big	FIPS 180-2 Change Notice 1
SHA-256	256	$< 2^{64}$	512	big	電子政府推奨暗号 FIPS 180-2 NESSIE Portfolio ISO/IEC 10118-3
SHA-384/512	384/512	$< 2^{128}$	1024	big	電子政府推奨暗号 FIPS 180-2 NESSIE Portfolio ISO/IEC 10118-3
Whirlpool	512	$< 2^{256}$	512	neutral	NESSIE Portfolio ISO/IEC 10118-3

注) 電子政府推奨暗号リストにおいて、RIPEMD-160, SHA-1 については「新たな電子政府用システムを構築する場合、より長いハッシュ値のものが使用できるのであれば、256 ビット以上のハッシュ関数を選択することが望ましい。ただし、公開鍵暗号での仕様上、利用すべきハッシュ関数が指定されている場合には、この限りではない。」という注釈がついている。

本章で用いる記号　表3.1に本章で用いる記号を定義する。

表 3.2: 記号の定義

記号	定義
$+$	ワード単位毎の算術加算
$\wedge$	ビット毎の論理積
$\vee$	ビット毎の論理和
$\oplus$	ビット毎の排他的論理和
$\neg x$	x のビット反転
$\mathrm{ROTR}^n(x)$	x の n ビット右巡回シフト
$\mathrm{ROTL}^n(x)$	x の n ビット左巡回シフト
$\mathrm{SHR}^n(x)$	x の n ビット右シフト

3.1 MD4

3.1.1 概要

MD4 は 1990 年に Rivest によって提案されたハッシュ関数である [R90]。Rivest は MD2, MD4, MD5 などの一連のハッシュ関数を提案したが、その後の専用ハッシュ関数の設計に特に大きな影響を与えたのが MD4 である。

MD4 は、ビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として 128 ビットのハッシュ値を出力する。演算は 32 ビット単位で行われ、全体は 3 ラウンド ×16 ステップで構成されている。

MD4 はソフトウェア、特に 32 ビットアーキテクチャで高速に実装できるよう、32 ビットワード演算を多用して設計されている。また、Intel 80xxx プロセッサを意識し、little-endian になっている[1]。

MD4 のアルゴリズムを記述した RFC1320 が 1992 年にリリースされたが [R92a]、1996 年に Dobbertin[D96a] により容易に衝突が発見できることが示されてから、利用は控えられている。

3.1.2 技術仕様

MD4 で使用される関数

MD4 では、32 ビットワードを入力変数および出力変数とする、以下の論理関数 $f_0, f_1, \ldots, f_{47}$ が用いられる。

$$f_t(x,y,z) = \begin{cases} (x \wedge y) \vee (\neg x \wedge z) & (0 \leq t \leq 15) \\ (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & (16 \leq t \leq 31) \\ x \oplus y \oplus z & (32 \leq t \leq 47) \end{cases}$$

$0 \leq t \leq 15$ の時、f_t は条件分岐関数 (もし x が真なら y、そうでなければ z)、$16 \leq t \leq 31$ の時、f_t は多数決関数、$32 \leq t \leq 47$ の時、f_t はパリティ関数となっており、全て x, y, z の 3 変数に関する対称関数となっている。

MD4 の前処理

1. 入力メッセージ M について、メッセージ長が 512 ビットの倍数になるように初期パディングされたメッセージ

$$M||1||0^k||l$$

[1] 当時、big-endian マシンの主流であった SUN Sparcstation は相対的に高速であったため、endian 変換のペナルティに耐えうるであろうという判断もあった [R90]。

を計算する。ただし、l は M のメッセージ長のバイナリ表現 (64 ビット[2])、k は $l+1+k \equiv 448 \pmod{512}$ を満たす正の最小値である。

2. 初期パディングされたメッセージは N 個の 512 ビット単位のブロック (block) $\{M^{(i)}\}_{i=1}^{N}$ に分割される。ただし、各々の $M^{(i)}$ は、16 個の 32 ビット長のワード

$$M^{(i)} = M_0^{(i)} || M_1^{(i)} || \cdots || M_{15}^{(i)}$$

からなる。

3. 初期値として

$$\begin{aligned} H_0^{(0)} &= \texttt{01234567} \\ H_1^{(0)} &= \texttt{89abcdef} \\ H_2^{(0)} &= \texttt{fedcba98} \\ H_3^{(0)} &= \texttt{76543210} \end{aligned}$$

を設定する。

MD4 のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \le i \le N$ の順に以下の手続き 1.〜6. (圧縮関数) (図 3.1 参照) を実行する。

1. メッセージブロックを X_j にコピーする。

$$X_j = M_j^{(i)} \quad (0 \le j \le 15)$$

2. 4 つの 32 ビット長のワード (A, B, C, D) を $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$\begin{aligned} A_0 &= H_0^{(i-1)} \\ B_0 &= H_1^{(i-1)} \\ C_0 &= H_2^{(i-1)} \\ D_0 &= H_3^{(i-1)} \end{aligned}$$

3. ラウンド 1 $(0 \le t \le 15)$

$$\begin{cases} A_{t+1} &= D_t \\ B_{t+1} &= \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_t + C_1) \\ C_{t+1} &= B_t \\ D_{t+1} &= C_t \end{cases}$$

[2]メッセージ長が 2^{64} より大きい場合、下位 64 ビットのみが使われ、上位ビットは無視される。

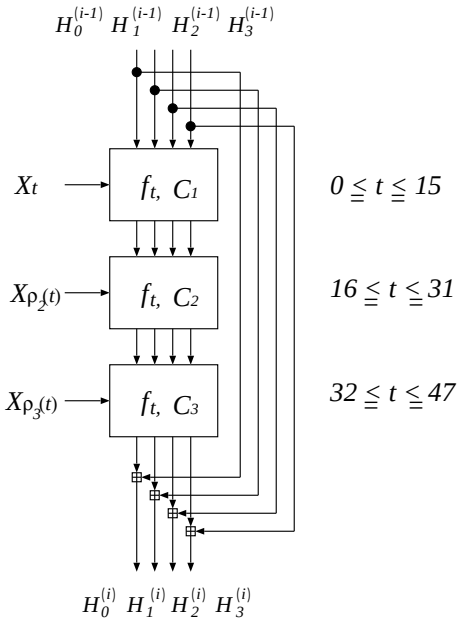

図 3.1: MD4 圧縮関数

4. ラウンド 2 $(16 \leq t \leq 31)$

$$\begin{cases} A_{t+1} & = D_t \\ B_{t+1} & = \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_{\rho_2(t)} + C_2) \\ C_{t+1} & = B_t \\ D_{t+1} & = C_t \end{cases}$$

5. ラウンド 3 $(32 \leq t \leq 47)$

$$\begin{cases} A_{t+1} & = D_t \\ B_{t+1} & = \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_{\rho_3(t)} + C_3) \\ C_{t+1} & = B_t \\ D_{t+1} & = C_t \end{cases}$$

各ラウンドにおけるローテーションビット数 $(s[t])$ は以下で定義される。

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ラウンド 1	3	7	11	19	3	7	11	19	3	7	11	19	3	7	11	19
ラウンド 2	3	5	9	13	3	5	9	13	3	5	9	13	3	5	9	13
ラウンド 3	3	9	11	15	3	9	11	15	3	9	11	15	3	9	11	15

ラウンド 2、ラウンド 3 においてメッセージ適用順序を示す関数 $\rho_2(t)$, $\rho_3(t)$ は以下で定義される。

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_2(t)$	0	4	8	12	1	5	9	13	2	6	10	14	3	7	11	15
$\rho_3(t)$	0	8	4	12	2	10	6	14	1	9	5	13	3	11	7	15

C_1, C_2, C_3 はラウンド定数と呼ばれる 32 ビットワードの定数で、下記の値の整数部分である。

	C_1	C_2	C_3
	0	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$
16 進表現	`00000000`	`5A827999`	`6ED9EBA1`

6. i 番目の中間ハッシュ値を

$$\begin{aligned} H_0^{(i)} &= H_0^{(i-1)} + A_{48} \\ H_1^{(i)} &= H_1^{(i-1)} + B_{48} \\ H_2^{(i)} &= H_2^{(i-1)} + C_{48} \\ H_3^{(i)} &= H_3^{(i-1)} + D_{48} \end{aligned}$$

で計算する。

上記手続き 1.〜 6.(圧縮関数) を N 回繰り返した最終的な 128 ビットの値

$$H^{(N)} = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)}$$

がメッセージ M のハッシュ値である。

3.2 MD5

3.2.1 概要

MD5 は MD4 の強度を高めた拡張アルゴリズムとして 1991 年に提案されたハッシュ関数である [R92b]。MD5 は、MD4 と同様に、ビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として 128 ビットのハッシュ値を出力する。演算は 32 ビット単位で行われ、全体は 4 ラウンド ×16 ステップで構成されている。MD4 と異なる点は、ラウンド数を 3 ラウンドから 4 ラウンドに増やした点、第 3 ラウンドのブール関数、メッセージワードの手順の変更、ラウンド定数の代わりに各演算に加算定数を追加した点である。

MD5 のアルゴリズムを記述した RFC1321 が 1992 年にリリースされ [R92b]、インターネット標準として広く利用されているハッシュ関数の一つであるが、2004 年に Wang ら [WFLY04] により衝突が発見されており、今後の利用は控えるべきと思われる。

3.2.2 技術仕様

MD5 で使用される関数

MD5 では、32 ビットワードを入力変数および出力変数とする、以下の論理関数 $f_0, f_1, \ldots, f_{63}$ が用いられる。

$$f_t(x,y,z) = \begin{cases} (x \wedge y) \vee (\neg x \wedge z) & (0 \leq t \leq 15) \\ (x \wedge z) \vee (y \wedge \neg z) & (16 \leq t \leq 31) \\ x \oplus y \oplus z & (32 \leq t \leq 47) \\ y \oplus (x \vee \neg z) & (48 \leq t \leq 63) \end{cases}$$

MD5 の前処理

1. 入力メッセージ M について、メッセージ長が 512 ビットの倍数になるように初期パディングされたメッセージ

   $$M||1||0^k||l$$

   を計算する。ただし、l は M のメッセージ長のバイナリ表現 (64 ビット[3])、k は $l + 1 + k \equiv 448 \pmod{512}$ を満たす正の最小値である。

[3] メッセージ長が 2^{64} より大きい場合、下位 64 ビットのみが使われ、上位ビットは無視される。

2. 初期パディングされたメッセージは N 個の 512 ビット単位のブロック (block) $\{M^{(i)}\}_{i=1}^{N}$ に分割される。ただし、各々の $M^{(i)}$ は、16 個の 32 ビット長のワード

$$M^{(i)} = M_0^{(i)}||M_1^{(i)}||\cdots||M_{15}^{(i)}$$

からなる。

3. 初期値として

$$\begin{aligned} H_0^{(0)} &= \texttt{01234567} \\ H_1^{(0)} &= \texttt{89abcdef} \\ H_2^{(0)} &= \texttt{fedcba98} \\ H_3^{(0)} &= \texttt{76543210} \end{aligned}$$

を設定する。

MD5 のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \leq i \leq N$ の順に以下の手続き 1.〜7. (圧縮関数) (図 3.2 参照) を実行する。

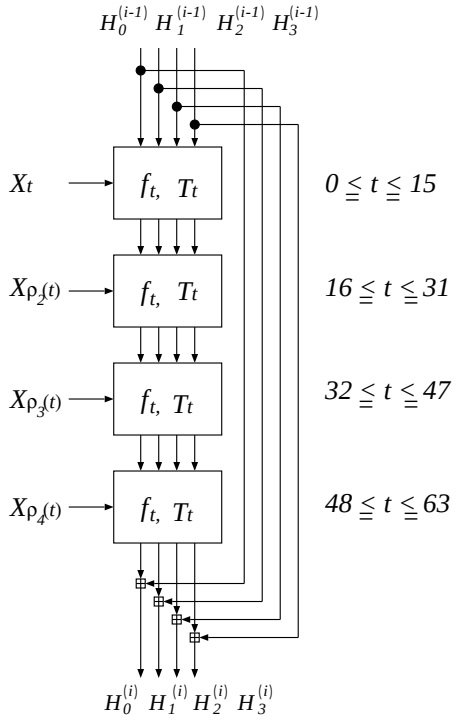

図 3.2: MD5 圧縮関数

1. メッセージブロックを X_j にコピーする。

$$X_j = M_j^{(i)} \quad (0 \leq j \leq 15)$$

2. 4 つの 32 ビット長のワード (A, B, C, D) を $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$\begin{aligned} A_0 &= H_0^{(i-1)} \\ B_0 &= H_1^{(i-1)} \\ C_0 &= H_2^{(i-1)} \\ D_0 &= H_3^{(i-1)} \end{aligned}$$

3. ラウンド 1 $(0 \leq t \leq 15)$

$$\begin{cases} A_{t+1} &= D_t \\ B_{t+1} &= \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_t + T_t) \\ C_{t+1} &= B_t \\ D_{t+1} &= C_t \end{cases}$$

4. ラウンド 2 $(16 \leq t \leq 31)$

$$\begin{cases} A_{t+1} &= D_t \\ B_{t+1} &= \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_{\rho_2(t)} + T_t) \\ C_{t+1} &= B_t \\ D_{t+1} &= C_t \end{cases}$$

5. ラウンド 3 $(32 \leq t \leq 47)$

$$\begin{cases} A_{t+1} &= D_t \\ B_{t+1} &= \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_{\rho_3(t)} + T_t) \\ C_{t+1} &= B_t \\ D_{t+1} &= C_t \end{cases}$$

6. ラウンド 4 $(48 \leq t \leq 63)$

$$\begin{cases} A_{t+1} &= D_t \\ B_{t+1} &= \mathrm{ROTL}^{s[t]}(A_t + f_t(B_t, C_t, D_t) + X_{\rho_4(t)} + T_t) \\ C_{t+1} &= B_t \\ D_{t+1} &= C_t \end{cases}$$

各ラウンドにおけるローテーションビット数 ($s[t]$) は以下で定義される。

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ラウンド 1	7	12	17	22	7	12	17	22	7	12	17	22	7	12	17	22
ラウンド 2	5	9	14	20	5	9	14	20	5	9	14	20	5	9	14	20
ラウンド 3	4	11	16	23	4	11	16	23	4	11	16	23	4	11	16	23
ラウンド 4	6	10	15	21	6	10	15	21	6	10	15	21	6	10	15	21

ラウンド 2、ラウンド 3、ラウンド 4 においてメッセージ適用順序を示す関数 $\rho_2(t)$, $\rho_3(t)$, $\rho_4(t)$ は以下で定義される。

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_2(t)$	1	6	11	0	5	10	15	4	9	14	3	8	13	2	7	12
$\rho_3(t)$	5	8	11	14	1	4	7	10	13	0	3	6	9	12	15	2
$\rho_4(t)$	0	7	14	5	12	3	10	1	8	15	6	13	4	11	2	9

ラウンド定数 T_t $(0 \leq t \leq 63)$ は正弦関数から導かれた定数である。

$$T_t = (4294967296 \times |\sin(t)|) \text{ の整数部分}$$

7. i 番目の中間ハッシュ値を

$$\begin{aligned} H_0^{(i)} &= H_0^{(i-1)} + A_{64} \\ H_1^{(i)} &= H_1^{(i-1)} + B_{64} \\ H_2^{(i)} &= H_2^{(i-1)} + C_{64} \\ H_3^{(i)} &= H_3^{(i-1)} + D_{64} \end{aligned}$$

で計算する。

上記手続き 1.〜 7.(圧縮関数) を N 回繰り返した最終的な 128 ビットの値

$$H^{(N)} = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)}$$

がメッセージ M のハッシュ値である。

3.3 RIPEMD

3.3.1 概要

RIPEMD は 1992 年にヨーロッパの RIPE (Race Integrity Primitive Evaluation) プロジェクトの成果の一つとして提案されたハッシュ関数である [RIPE92]。RIPEMD は、MD4、MD5 と同じく、ビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として、128 ビットのハッシュ値を出力する。

RIPEMD は den Boer と Bosselaer による MD4 の圧縮関数の最終 2 ラウンドに対する攻撃 [DB92] を考慮して改良されたアルゴリズムで、MD4 圧縮関数に変更を加えた関数を 2 並列に実行する構成となっている。演算は 32 ビット単位で行われ、全体は 3 ラウンド ×16 ステップを 2 並列に実行する構成となっている。

MD4 と異なる点は、以下の通りである。

- 各ステップでのローテーションビット数及びメッセージワードの適用順序が MD4 と異なる。
- 左ライン、右ラインの 2 並列処理は、ラウンド定数のみが異なる。
- 圧縮関数の最後で左右のラインの結果が加算される。

RIPEMD は、1995 年に Dobbertin により、3 ラウンド中の最初の 2 ラウンド及び最終 2 ラウンドに対する衝突攻撃が発表され、RIPEMD アルゴリズム自体が標準化されることはなかったが、設計指針は RIPEMD-160, RIPEMD-128 に引き継がれている。

3.3.2 技術仕様

RIPEMD で使用される関数

RIPEMD では、MD4 と同じ以下の論理関数 $f_0, f_1, \ldots, f_{47}$ が用いられる。

$$f_t(x,y,z) = \begin{cases} (x \wedge y) \vee (\neg x \wedge z) & (0 \le t \le 15) \\ (x \wedge y) \vee (x \wedge z) \vee (y \wedge z) & (16 \le t \le 31) \\ x \oplus y \oplus z & (32 \le t \le 47) \end{cases}$$

RIPEMD の前処理

1. 入力メッセージ M について、メッセージ長が 512 ビットの倍数になるように初期パディングされたメッセージ

$$M||1||0^k||l$$

を計算する。ただし、l は M のメッセージ長のバイナリ表現 (64 ビット[4])、k は $l+1+k \equiv 448 \pmod{512}$ を満たす正の最小値である。

2. 初期パディングされたメッセージは N 個の 512 ビット単位のブロック (block) $\{M^{(i)}\}_{i=1}^{N}$ に分割される。ただし、各々の $M^{(i)}$ は、16 個の 32 ビット長のワード

$$M^{(i)} = M_0^{(i)} || M_1^{(i)} || \cdots || M_{15}^{(i)}$$

からなる。

3. 初期値として

$$\begin{aligned} H_0^{(0)} &= \texttt{67452301} \\ H_1^{(0)} &= \texttt{efcdab89} \\ H_2^{(0)} &= \texttt{98badcfe} \\ H_3^{(0)} &= \texttt{10325476} \end{aligned}$$

を設定する。

RIPEMD のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \le i \le N$ の順に以下の手続き 1.〜6. (圧縮関数) (図 3.3 参照) を実行する。

1. メッセージブロックを X_j にコピーする。

$$X_j = M_j^{(i)} \quad (0 \le j \le 15)$$

2. 左右のラインそれぞれについて、4 つの 32 ビット長のワード $(A_{Lt}, B_{Lt}, C_{Lt}, D_{Lt})$, $(A_{Rt}, B_{Rt}, C_{Rt}, D_{Rt})$, $(0 \le t \le 48)$ がハッシュ値を計算するためのバッファとして用いられる。これらを $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$\begin{aligned} A_{L0} &= A_{R0} = H_0^{(i-1)} \\ B_{L0} &= B_{R0} = H_1^{(i-1)} \\ C_{L0} &= C_{R0} = H_2^{(i-1)} \\ D_{L0} &= D_{R0} = H_3^{(i-1)} \end{aligned}$$

[4] メッセージ長が 2^{64} より大きい場合、下位 64 ビットのみが使われ、上位ビットは無視される。

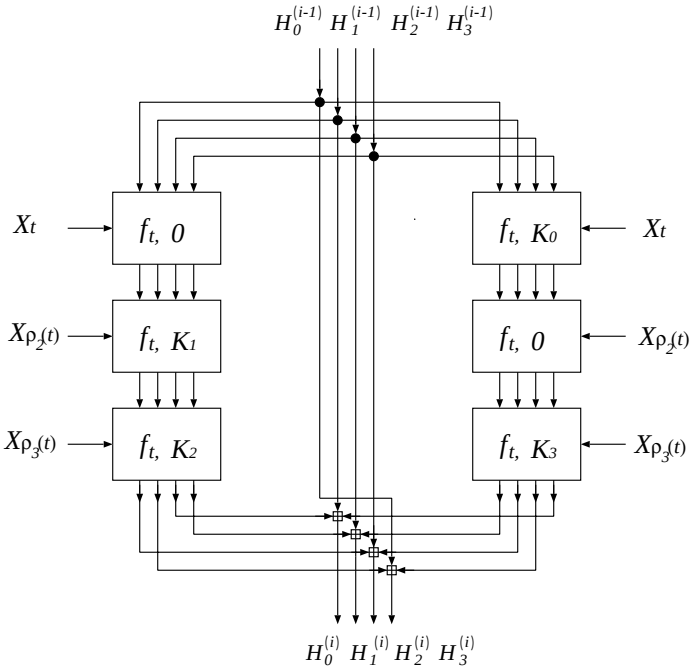

図 3.3: RIPEMD 圧縮関数

3. ラウンド 1 $(0 \leq t \leq 15)$

$$\text{左ライン} \begin{cases} A_{Lt+1} &= D_{Lt} \\ B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_t(B_{Lt}, C_{Lt}, D_{Lt}) + X_t) \\ C_{Lt+1} &= B_{Lt} \\ D_{Lt+1} &= C_{Lt} \end{cases}$$

$$\text{右ライン} \begin{cases} A_{Rt+1} &= D_{Rt} \\ B_{Rt+1} &= \mathrm{ROTL}^{s[t]}(A_{Rt} + f_t(B_{Rt}, C_{Rt}, D_{Rt}) + X_t + K_0) \\ C_{Rt+1} &= B_{Rt} \\ D_{Rt+1} &= C_{Rt} \end{cases}$$

4. ラウンド 2 $(16 \leq t \leq 31)$

$$\text{左ライン} \begin{cases} A_{Lt+1} &= D_{Lt} \\ B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_t(B_{Lt}, C_{Lt}, D_{Lt}) + X_{\rho_2(t)} + K_1) \\ C_{Lt+1} &= B_{Lt} \\ D_{Lt+1} &= C_{Lt} \end{cases}$$

$$\text{右ライン} \begin{cases} A_{Rt+1} &= D_{Rt} \\ B_{Rt+1} &= \mathrm{ROTL}^{s[t]}(A_{Rt} + f_t(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\rho_2(t)}) \\ C_{Rt+1} &= B_{Rt} \\ D_{Rt+1} &= C_{Rt} \end{cases}$$

5. ラウンド3 ($32 \leq t \leq 47$)

$$\text{左ライン}\begin{cases} A_{Lt+1} &= D_{Lt} \\ B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_t(B_{Lt}, C_{Lt}, D_{Lt}) + X_{\rho_3(t)} + K_2) \\ C_{Lt+1} &= B_{Lt} \\ D_{Lt+1} &= C_{Lt} \end{cases}$$

$$\text{右ライン}\begin{cases} A_{Rt+1} &= D_{Rt} \\ B_{Rt+1} &= \mathrm{ROTL}^{s[t]}(A_{Rt} + f_t(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\rho_3(t)} + K_3) \\ C_{Rt+1} &= B_{Rt} \\ D_{Rt+1} &= C_{Rt} \end{cases}$$

各ラウンドにおけるローテーションビット数 ($s[t]$) は以下で定義される。

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ラウンド1	11	14	15	12	5	8	7	9	11	13	14	15	6	7	9	8
ラウンド2	7	6	8	13	11	9	7	15	7	12	15	9	7	11	13	12
ラウンド3	11	13	14	7	14	9	13	15	6	8	13	6	12	5	7	5

ラウンド2、ラウンド3においてメッセージ適用順序を示す関数 $\rho_2(t)$, $\rho_3(t)$ は以下で定義される。

t	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho_2(t)$	7	4	13	1	10	6	15	3	12	0	9	5	14	2	11	8
$\rho_3(t)$	3	10	2	4	9	15	8	1	14	7	0	6	11	13	5	12

K_0, K_1, K_2, K_3 はラウンド定数と呼ばれる 32 ビットワードの定数で、下記の値の整数部分である。

	K_0	K_1	K_2	K_3
	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt[3]{3}$
16進表現	`50a28be6`	`5A827999`	`6ED9EBA1`	`5C4DD124`

6. i 番目の中間ハッシュ値を

$$\begin{aligned} H_0^{(i)} &= H_1^{(i-1)} + C_{L48} + D_{R48} \\ H_1^{(i)} &= H_2^{(i-1)} + D_{L48} + A_{R48} \\ H_2^{(i)} &= H_3^{(i-1)} + A_{L48} + B_{R48} \\ H_3^{(i)} &= H_0^{(i-1)} + B_{L48} + C_{R48} \end{aligned}$$

で計算する。

上記手続き 1.〜 6.(圧縮関数) を N 回繰り返した最終的な 128 ビットの値

$$H^{(N)} = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)}$$

がメッセージ M のハッシュ値である。

3.4 RIPEMD-160

3.4.1 概要

RIPEMD-160 は、RIPEMD の強度を高めたアルゴリズムとして、RIPEMD-128 とともに 1996 年に Dobbertin, Bosselaers, Preneel により提案されたハッシュ関数である [DBP96]。

RIPEMD-160 は、ビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として 160 ビットのハッシュ値を出力する。

RIPEMD-160 は 2 つのほぼ同じ形をした関数を 2 並列に実行する。2 つの関数は右ラインおよび左ラインと呼ばれ、各々5 ラウンド 80 ステップで構成される。

RIPEMD-160 は国際規格 ISO/IEC 10118-3 [ISO/IEC10118-3] 及び電子政府推奨暗号リスト [CRYPTREC03] に採用されている。

3.4.2 技術仕様

RIPEMD-160 で使用される関数

$$
\begin{aligned}
f_1(x,y,z) &= x \oplus y \oplus z \\
f_2(x,y,z) &= (x \wedge y) \vee (\neg x \wedge z) \\
f_3(x,y,z) &= (x \wedge \neg y) \oplus z \\
f_4(x,y,z) &= (x \wedge y) \vee (y \wedge \neg z) \\
f_5(x,y,z) &= x \oplus (y \vee \neg z)
\end{aligned}
$$

これらのブール関数の適用順序は次の通りである。

ライン	ラウンド 1	ラウンド 2	ラウンド 3	ラウンド 4	ラウンド 5
左	f_1	f_2	f_3	f_4	f_5
右	f_5	f_4	f_3	f_2	f_1

RIPEMD の前処理

1. 入力メッセージ M について、メッセージ長が 512 ビットの倍数になるように初期パディングされたメッセージ

   $$M||1||0^k||l$$

   を計算する。ただし、l は M のメッセージ長のバイナリ表現 (64 ビット[5])、k は $l + 1 + k \equiv 448 \pmod{512}$ を満たす正の最小値である。

[5] メッセージ長が 2^{64} より大きい場合、下位 64 ビットのみが使われ、上位ビットは無視される。

2. 初期パディングされたメッセージは N 個の 512 ビット単位のブロック (block) $\{M^{(i)}\}_{i=1}^{N}$ に分割される。ただし、各々の $M^{(i)}$ は、16 個の 32 ビット長のワード

$$M^{(i)} = M_0^{(i)}||M_1^{(i)}||\cdots||M_{15}^{(i)}$$

からなる。

3. 初期値として

$$\begin{aligned} H_0^{(0)} &= \texttt{67452301} \\ H_1^{(0)} &= \texttt{efcdab89} \\ H_2^{(0)} &= \texttt{98badcfe} \\ H_3^{(0)} &= \texttt{10325476} \\ H_4^{(0)} &= \texttt{c3d2e1f0} \end{aligned}$$

を設定する。

RIPEMD-160 のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \le i \le N$ の順に以下の手続き 1.〜8. (圧縮関数) (図 3.4 参照) を実行する。

1. メッセージブロックを X_j にコピーする。

$$X_j = M_j^{(i)} \quad (0 \le j \le 15)$$

2. 左右のラインそれぞれについて、4 つの 32 ビット長のワード $(A_{Lt}, B_{Lt}, C_{Lt}, D_{Lt})$, $(A_{Rt}, B_{Rt}, C_{Rt}, D_{Rt})$, $(0 \le t \le 48)$ がハッシュ値を計算するためのバッファとして用いられる。これらを $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$\begin{aligned} A_{L0} &= A_{R0} = H_0^{(i-1)} \\ B_{L0} &= B_{R0} = H_1^{(i-1)} \\ C_{L0} &= C_{R0} = H_2^{(i-1)} \\ D_{L0} &= D_{R0} = H_3^{(i-1)} \\ E_{L0} &= E_{R0} = H_4^{(i-1)} \end{aligned}$$

3. ラウンド 1 $(1 \le t \le 16)$

$$\text{左ライン} \begin{cases} A_{Lt+1} &= E_{Lt} \\ B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_1(B_{Lt}, C_{Lt}, D_{Lt}) + X_t + K_1) + E_{Lt} \\ C_{Lt+1} &= B_{Lt} \\ D_{Lt+1} &= \mathrm{ROTL}^{10}(C_{Lt}) \\ E_{Lt+1} &= D_{Lt} \end{cases}$$

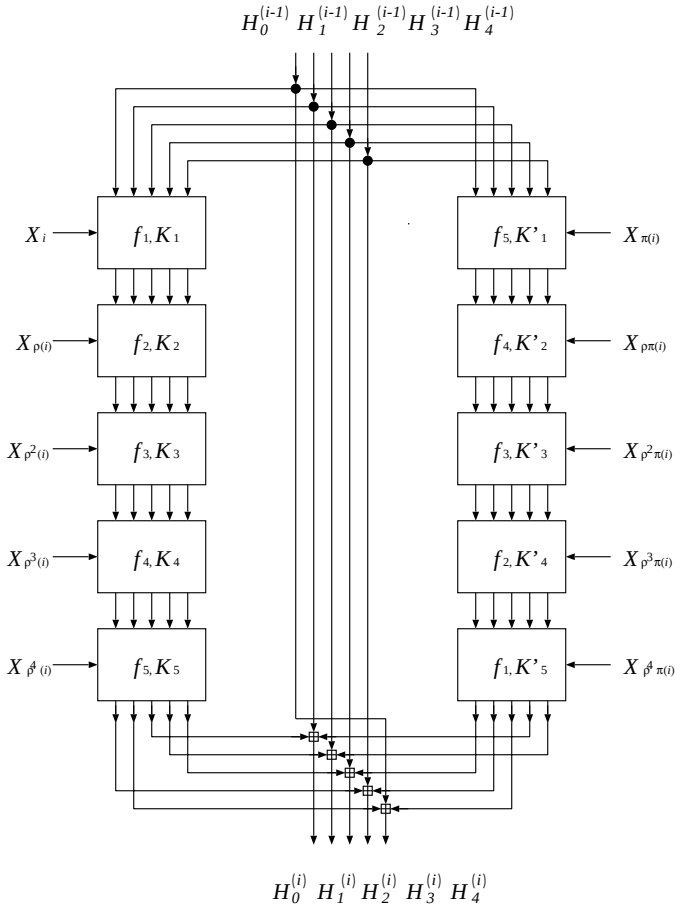

図 3.4: RIPEMD-160 圧縮関数

$$
\text{右ライン}\begin{cases}
A_{Rt+1} &= E_{Rt} \\
B_{Rt+1} &= \mathrm{ROTL}^{s'[t]}(A_{Rt} + f_5(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\pi(t)} + K'_1) + E_{Rt} \\
C_{Rt+1} &= B_{Rt} \\
D_{Rt+1} &= \mathrm{ROTL}^{10}(C_{Rt}) \\
E_{Rt+1} &= D_{Rt}
\end{cases}
$$

4. ラウンド 2 ($17 \leq t \leq 32$)

$$
\text{左ライン}\begin{cases}
A_{Lt+1} &= E_{Lt} \\
B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_2(B_{Lt}, C_{Lt}, D_{Lt}) + X_{\rho(t)} + K_2) + E_{Lt} \\
C_{Lt+1} &= B_{Lt} \\
D_{Lt+1} &= \mathrm{ROTL}^{10}(C_{Lt}) \\
E_{Lt+1} &= D_{Lt}
\end{cases}
$$

$$
\text{右ライン}\begin{cases}
A_{Rt+1} &= E_{Rt} \\
B_{Rt+1} &= \mathrm{ROTL}^{s'[t]}(A_{Rt} + f_4(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\rho\pi(t)} + K'_2) + E_{Rt} \\
C_{Rt+1} &= B_{Rt} \\
D_{Rt+1} &= \mathrm{ROTL}^{10}(C_{Rt}) \\
E_{Rt+1} &= D_{Rt}
\end{cases}
$$

5. ラウンド 3 ($33 \leq t \leq 48$)

$$
\text{左ライン}\begin{cases}
A_{Lt+1} &= E_{Lt} \\
B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_3(B_{Lt}, C_{Lt}, D_{Lt}) + X_{\rho^2(t)} + K_3) + E_{Lt} \\
C_{Lt+1} &= B_{Lt} \\
D_{Lt+1} &= \mathrm{ROTL}^{10}(C_{Lt}) \\
E_{Lt+1} &= D_{Lt}
\end{cases}
$$

$$
\text{右ライン}\begin{cases}
A_{Rt+1} &= E_{Rt} \\
B_{Rt+1} &= \mathrm{ROTL}^{s'[t]}(A_{Rt} + f_3(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\rho^2\pi(t)} + K'_3) + E_{Rt} \\
C_{Rt+1} &= B_{Rt} \\
D_{Rt+1} &= \mathrm{ROTL}^{10}(C_{Rt}) \\
E_{Rt+1} &= D_{Rt}
\end{cases}
$$

6. ラウンド 4 ($49 \leq t \leq 64$)

$$
\text{左ライン}\begin{cases}
A_{Lt+1} &= E_{Lt} \\
B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_4(B_{Lt}, C_{Lt}, D_{Lt}) + X_{\rho^3(t)} + K_4) + E_{Lt} \\
C_{Lt+1} &= B_{Lt} \\
D_{Lt+1} &= \mathrm{ROTL}^{10}(C_{Lt}) \\
E_{Lt+1} &= D_{Lt}
\end{cases}
$$

$$
\text{右ライン}\begin{cases}
A_{Rt+1} &= E_{Rt} \\
B_{Rt+1} &= \mathrm{ROTL}^{s'[t]}(A_{Rt} + f_2(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\rho^3\pi(t)} + K'_4) + E_{Rt} \\
C_{Rt+1} &= B_{Rt} \\
D_{Rt+1} &= \mathrm{ROTL}^{10}(C_{Rt}) \\
E_{Rt+1} &= D_{Rt}
\end{cases}
$$

7. ラウンド 5 ($65 \leq t \leq 80$)

$$
\text{左ライン}\begin{cases}
A_{Lt+1} &= E_{Lt} \\
B_{Lt+1} &= \mathrm{ROTL}^{s[t]}(A_{Lt} + f_5(B_{Lt}, C_{Lt}, D_{Lt}) + X_{\rho^4(t)} + K_5) + E_{Lt} \\
C_{Lt+1} &= B_{Lt} \\
D_{Lt+1} &= \mathrm{ROTL}^{10}(C_{Lt}) \\
E_{Lt+1} &= D_{Lt}
\end{cases}
$$

$$
\text{右ライン}\begin{cases}
A_{Rt+1} &= E_{Rt} \\
B_{Rt+1} &= \mathrm{ROTL}^{s'[t]}(A_{Rt} + f_1(B_{Rt}, C_{Rt}, D_{Rt}) + X_{\rho^4\pi(t)} + K'_5) + E_{Rt} \\
C_{Rt+1} &= B_{Rt} \\
D_{Rt+1} &= \mathrm{ROTL}^{10}(C_{Rt}) \\
E_{Rt+1} &= D_{Rt}
\end{cases}
$$

但し、メッセージワードの適用順序に対し、以下の置換 ρ と π が定義されている。

置換 ρ の定義

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\rho(i)$	7	4	13	1	10	6	15	3	12	0	9	5	2	14	11	8

置換 π の定義

$$\pi(i) = 9i + 5 \pmod{16}$$

メッセージワードの適用順序は次のようになる。

ライン	ラウンド 1	ラウンド 2	ラウンド 3	ラウンド 4	ラウンド 5
左	恒等変換	ρ	ρ^2	ρ^3	ρ^4
右	π	$\rho\pi$	$\rho^2\pi$	$\rho^3\pi$	$\rho^4\pi$

ラウンド定数は下記の値の整数部分である。

左ライン	K_1	K_2	K_3	K_4	K_5
	0	$2^{30} \cdot \sqrt{2}$	$2^{30} \cdot \sqrt{3}$	$2^{30} \cdot \sqrt{5}$	$2^{30} \cdot \sqrt{7}$
16 進表現	00000000	5A827999	6ED9EBA1	8F1BBCDC	A953FD4E
右ライン	K'_1	K'_2	K'_3	K'_4	K'_5
	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt[3]{3}$	$2^{30} \cdot \sqrt[3]{5}$	$2^{30} \cdot \sqrt{7}$	0
16 進表現	50A28BE6	5C4DD124	6D703EF3	7A6D76E9	00000000

また、ステップ関数で用いられる左巡回シフト量 $s[t], s'[t]$ はあらかじめ定められている。

8. i 番目の中間ハッシュ値を

$$
\begin{aligned}
H_0^{(i)} &= H_1^{(i-1)} + C_{L80} + D_{R80} \\
H_1^{(i)} &= H_2^{(i-1)} + D_{L80} + E_{R80} \\
H_2^{(i)} &= H_3^{(i-1)} + E_{L80} + A_{R80} \\
H_3^{(i)} &= H_4^{(i-1)} + A_{L80} + B_{R80} \\
H_4^{(i)} &= H_0^{(i-1)} + B_{L80} + C_{R80}
\end{aligned}
$$

で計算する。

上記手続き 1.〜 8.(圧縮関数) を N 回繰り返した最終的な 160 ビットの値

$$H^{(N)} = H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}$$

がメッセージ M のハッシュ値である。

3.5 RIPEMD-128

3.5.1 概要

RIPEMD-128 は、RIPEMD の強度を高めたアルゴリズムとして 1996 年に Dobbertin, Bosselaers, Preneel により提案されたハッシュ関数である [DBP96]。

RIPEMD-128 は、ビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として 128 ビットのハッシュ値を出力する。

RIPEMD-128 は、RIPEMD-160 をベースに設計されており、(A, B, C, D) の 4 変数のみ使うこと、5 ラウンドのうち 4 ラウンドのみを使うこと、各ラウンドで用いるブール関数とラウンド定数のみが異なっている。

RIPEMD-128 は国際規格 ISO/IEC 10118-3 [ISO/IEC10118-3] に採用されている。

3.5.2 技術仕様

RIPEMD-128 は、RIPEMD-160 をベースに設計されており、(A, B, C, D) の 4 変数のみ使うこと、5 ラウンドのうち 4 ラウンドのみを使うこと、各ラウンドで用いるブール関数とラウンド定数のみが異なっている。

ブール関数の適用順序は次の通りである。

ライン	ラウンド 1	ラウンド 2	ラウンド 3	ラウンド 4
左	f_1	f_2	f_3	f_4
右	f_4	f_3	f_2	f_1

ラウンド定数は以下の値の整数部分である。

ライン	ラウンド 1	ラウンド 2	ラウンド 3	ラウンド 4
左	0	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt[3]{3}$	$2^{30} \cdot \sqrt[3]{5}$
右	$2^{30} \cdot \sqrt[3]{2}$	$2^{30} \cdot \sqrt[3]{3}$	$2^{30} \cdot \sqrt[3]{5}$	0

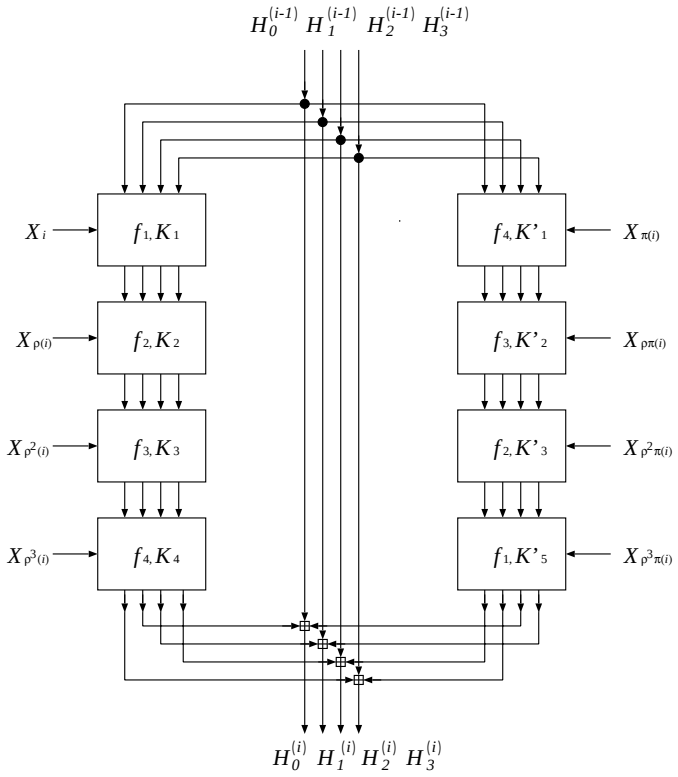

図 3.5: RIPEMD-128 圧縮関数

3.6 SHA-1

3.6.1 概要

SHS (Secure Hash Standard) は米国商務省 技術標準機関 NIST (National Institute of Standards and Technology) が制定したハッシュ関数の標準規格である。SHA (Secure Hash Algorithm) は MD4 及び MD5 をベースに米国国家安全局 NSA (National Security Agency) により設計され、1993 年に NIST により米国連邦政府情報処理規格 FIPS (Federal Information Processing Standard) 180 [FIPS180] として制定された。

SHA はビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として、160 ビットのハッシュ値を出力する。

その後、SHA (SHA-0) のメッセージスケジュール関数のみに若干の仕様変更 (1 ビットローテーションの追加) がなされ、このアルゴリズム (SHA-1) が、SHA (SHA-0) を置き換える形で 1995 年に FIPS180-1 [FIPS180-1] として制定された。

3.6.2 技術仕様

SHA-1 で使用される関数

SHA-1 では、32 ビットワードを入力変数および出力変数とする、以下の論理関数 $f_0, f_1, \ldots, f_{79}$ が用いられる。

$$f_t(x,y,z) = \begin{cases} \mathrm{Ch}(x,y,z) = (x \wedge y) \oplus (\neg x \wedge z) & (0 \le t \le 19) \\ \mathrm{Parity}(x,y,z) = x \oplus y \oplus z & (20 \le t \le 39) \\ \mathrm{Maj}(x,y,z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) & (40 \le t \le 59) \\ \mathrm{Parity}(x,y,z) = x \oplus y \oplus z & (60 \le t \le 79) \end{cases}$$

SHA-1 の前処理

1. 入力メッセージ M について、メッセージ長が 512 ビットの倍数になるように初期パディングされたメッセージ

   $$M||1||0^k||l$$

   を計算する。ただし、l は M のメッセージ長のバイナリ表現 (64 ビット)、k は $l + 1 + k \equiv 448 \pmod{512}$ を満たす正の最小値である。

2. 初期パディングされたメッセージは N 個の 512 ビット単位のブロック (block) $\{M^{(i)}\}_{i=1}^{N}$ に分割される。ただし、各々の $M^{(i)}$ は、16 個の 32 ビット長のワード

   $$M^{(i)} = M_0^{(i)}||M_1^{(i)}||\cdots||M_{15}^{(i)}$$

   からなる。

3. 初期値として

$$\begin{aligned} H_0^{(0)} &= \texttt{67452301} \\ H_1^{(0)} &= \texttt{efcdab89} \\ H_2^{(0)} &= \texttt{98badcfe} \\ H_3^{(0)} &= \texttt{10325476} \\ H_4^{(0)} &= \texttt{c3d2e1f0} \end{aligned}$$

を設定する。

SHA-1 のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \leq i \leq N$ の順に以下の手続きを実行する。

1. 次式で定義する SHA-1 メッセージスケジュール関数を用いて拡張メッセージ W_t を計算する。

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \mathrm{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) & 16 \leq t \leq 79 \end{cases}$$

2. 5 個のバッファ変数を $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$\begin{aligned} a_0 &= H_0^{(i-1)} \\ b_0 &= H_1^{(i-1)} \\ c_0 &= H_2^{(i-1)} \\ d_0 &= H_3^{(i-1)} \\ e_0 &= H_4^{(i-1)} \end{aligned}$$

3. $0 \leq t \leq 79$ に対して以下の計算を繰り返す。

$$\begin{cases} T &= \mathrm{ROTL}^5(a_t) + f_t(b_t, c_t, d_t) + e_t + K_t + W_t \\ e_{t+1} &= d_t \\ d_{t+1} &= c_t \\ c_{t+1} &= \mathrm{ROTL}^{30}(b_t) \\ b_{t+1} &= a_t \\ a_{t+1} &= T \end{cases}$$

ただし、$K_t (0 \leq t \leq 79)$ は 32 ビットワードの定数である。

4. i 番目の中間ハッシュ値を

$$\begin{aligned} H_0^{(i)} &= H_0^{(i-1)} + a_{80} \\ H_1^{(i)} &= H_1^{(i-1)} + b_{80} \\ H_2^{(i)} &= H_2^{(i-1)} + c_{80} \\ H_3^{(i)} &= H_3^{(i-1)} + d_{80} \\ H_4^{(i)} &= H_4^{(i-1)} + e_{80} \end{aligned}$$

で計算する。

上記手続き 1.〜 4. を N 回繰り返した最終的な 160 ビットの値

$$H^{(N)} = H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}$$

がメッセージ M のハッシュ値である。

3.7 SHA-256

3.7.1 概要

SHA-256 は、SHA-384, SHA-512 とともに 2000 年に米国商務省技術標準機関 NIST により提案され、2002 年に FIPS180-2 [FIPS180-2] として制定された。

SHA-256 は、ビット長が 512 ビットの倍数になるようにパディングされた任意のメッセージを入力として 256 ビットのハッシュ値を出力する。

SHA-256 は ISO/IEC 10118-3 [ISO/IEC10118-3] の国際規格にも採用されている。

3.7.2 技術仕様

SHA-256 で使用される関数

SHA-256 では、32 ビットワードを入力変数および出力変数とする、以下の 6 種類の論理関数が用いられる。

$$\begin{cases} \mathrm{Ch}(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ \mathrm{Maj}(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\ \Sigma_0^{256}(x) &= \mathrm{ROTR}^2(x) \oplus \mathrm{ROTR}^{13}(x) \oplus \mathrm{ROTR}^{22}(x) \\ \Sigma_1^{256}(x) &= \mathrm{ROTR}^6(x) \oplus \mathrm{ROTR}^{11}(x) \oplus \mathrm{ROTR}^{25}(x) \\ \sigma_0^{256}(x) &= \mathrm{ROTR}^7(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{SHR}^3(x) \\ \sigma_1^{256}(x) &= \mathrm{ROTR}^{17}(x) \oplus \mathrm{ROTR}^{19}(x) \oplus \mathrm{SHR}^{10}(x) \end{cases}$$

SHA-256 の前処理

1. 入力メッセージ M について、メッセージ長が 512 ビットの倍数になるように初期パディングされたメッセージ

   $$M||1||0^k||l$$

   を計算する。ただし、l は M のメッセージ長のバイナリ表現 (64 ビット)、k は $l+1+k \equiv 448 \pmod{512}$ を満たす正の最小値である。

2. 初期パディングされたメッセージは N 個の 512 ビット単位のブロック $\{M^{(i)}\}_{i=1}^{N}$ に分割される。ただし、各々の $M^{(i)}$ は、16 個の 32 ビット長のワード

   $$M^{(i)} = M_0^{(i)}||M_1^{(i)}||\cdots||M_{15}^{(i)}$$

   からなる。

3. 初期値として

$$\begin{aligned} H_0^{(0)} &= \texttt{6a09e667} \\ H_1^{(0)} &= \texttt{bb67ae85} \\ H_2^{(0)} &= \texttt{3c6ef372} \\ H_3^{(0)} &= \texttt{a54ff53a} \\ H_4^{(0)} &= \texttt{510e527f} \\ H_5^{(0)} &= \texttt{9b05688c} \\ H_6^{(0)} &= \texttt{1f83d9ab} \\ H_7^{(0)} &= \texttt{5be0cd19} \end{aligned}$$

を設定する。

SHA-256 のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \leq i \leq N$ の順に以下を実行する。

1. 次式で定義する SHA-256 メッセージスケジュール関数を用いて拡張メッセージ W_t を計算する。

$$W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{256}(W_{t-2}) + W_{t-7} + \sigma_0^{256}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 63 \end{cases}$$

2. 8 個のバッファ変数を $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$\begin{aligned} a_0 &= H_0^{(i-1)} \\ b_0 &= H_1^{(i-1)} \\ c_0 &= H_2^{(i-1)} \\ d_0 &= H_3^{(i-1)} \\ e_0 &= H_4^{(i-1)} \\ f_0 &= H_5^{(i-1)} \\ g_0 &= H_6^{(i-1)} \\ h_0 &= H_7^{(i-1)} \end{aligned}$$

3. $0 \leq t \leq 63$ に対して以下の計算を繰り返す。

$$\begin{cases} T_1 &= h_t + \Sigma_1^{256}(e_t) + \mathrm{Ch}(e_t, f_t, g_t) + K_t^{256} + W_t \\ T_2 &= \Sigma_0^{256}(a_t) + \mathrm{Maj}(a_t, b_t, c_t) \\ h_{t+1} &= g_t \\ g_{t+1} &= f_t \\ f_{t+1} &= e_t \\ e_{t+1} &= d_t + T_1 \\ d_{t+1} &= c_t \\ c_{t+1} &= b_t \\ b_{t+1} &= a_t \\ a_{t+1} &= T_1 + T_2 \end{cases}$$

ただし、K_t^{256} は 32 ビットワードの定数 (FIPS PUB 180-2 参照) である。

4. i 番目の中間ハッシュ値を

$$\begin{aligned} H_0^{(i)} &= H_0^{(i-1)} + a_{64} \\ H_1^{(i)} &= H_1^{(i-1)} + b_{64} \\ H_2^{(i)} &= H_2^{(i-1)} + c_{64} \\ H_3^{(i)} &= H_3^{(i-1)} + d_{64} \\ H_4^{(i)} &= H_4^{(i-1)} + e_{64} \\ H_5^{(i)} &= H_5^{(i-1)} + f_{64} \\ H_6^{(i)} &= H_6^{(i-1)} + g_{64} \\ H_7^{(i)} &= H_7^{(i-1)} + h_{64} \end{aligned}$$

で計算する。

上記手続き 1.〜 4. を N 回繰り返した最終的な 256 ビットの値

$$H^{(N)} = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)} || H_6^{(N)} || H_7^{(N)}$$

がメッセージ M のハッシュ値である。

3.8 SHA-224

3.8.1 概要

SHA-224 は、2004 年 2 月に FIPS 180-2, Change Notice 1 [FIPS180-2a] に追加されたハッシュ長 224 ビットのハッシュ関数である。それまで FIPS 180-2 には、RSA-2048 と同等のセキュリティレベルに対応する 112 ビットセキュリティのハッシュ関数がなかった。

SHA-224 は、メッセージ長が 2^{64} ビット未満のメッセージを入力として 224 ビットのハッシュ値を出力する。演算は SHA-256 と同じく 32 ビット単位で行われる。

3.8.2 技術仕様

SHA-224 は以下の 2 点を除き、SHA-256 と同じ仕様である。

1. 初期値 $H^{(0)}$ を以下の値に設定する。

$$\begin{aligned} H_0^{(0)} &= \texttt{c1059ed8} \\ H_1^{(0)} &= \texttt{367cd507} \\ H_2^{(0)} &= \texttt{3070dd17} \\ H_3^{(0)} &= \texttt{f70e5939} \\ H_4^{(0)} &= \texttt{ffc00b31} \\ H_5^{(0)} &= \texttt{68581511} \\ H_6^{(0)} &= \texttt{64f98fa7} \\ H_7^{(0)} &= \texttt{befa4fa4} \end{aligned}$$

2. 圧縮関数を N 回繰り返した最終的な 256 ビットの値 $H^{(N)}$ の左 224 ビットの値

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}||H_6^{(N)}$$

を SHA-224 でのメッセージ M のハッシュ値とする。

3.9 SHA-512

3.9.1 概要

SHA-512は、SHA-256, SHA-384とともに、2000年に米国商務省 技術標準機関NISTにより提案され、2002年にFIPS180-2 [FIPS180-2] として制定された。

SHA-512のアルゴリズムは、SHA-256のワード長32ビットを64ビットに変更し、メッセージスケジュール関数の繰り返し回数を増やしたものである。

SHA-512は、ビット長が1024ビットの倍数になるようにパディングされた任意のメッセージを入力として512ビットのハッシュ値を出力する。

SHA-512はISO/IEC 10118-3 [ISO/IEC10118-3] の国際規格にも採用されている。

3.9.2 技術仕様

SHA-512で使用される関数

SHA-512では、64ビットワードを入力変数および出力変数とする、以下の6種類の論理関数が用いられる。

$$\begin{cases} \mathrm{Ch}(x,y,z) &= (x \wedge y) \oplus (\neg x \wedge z) \\ \mathrm{Maj}(x,y,z) &= (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z) \\ \Sigma_0^{512}(x) &= \mathrm{ROTR}^{28}(x) \oplus \mathrm{ROTR}^{34}(x) \oplus \mathrm{ROTR}^{39}(x) \\ \Sigma_1^{512}(x) &= \mathrm{ROTR}^{14}(x) \oplus \mathrm{ROTR}^{18}(x) \oplus \mathrm{ROTR}^{41}(x) \\ \sigma_0^{512}(x) &= \mathrm{ROTR}^{1}(x) \oplus \mathrm{ROTR}^{8}(x) \oplus \mathrm{SHR}^{7}(x) \\ \sigma_1^{512}(x) &= \mathrm{ROTR}^{19}(x) \oplus \mathrm{ROTR}^{61}(x) \oplus \mathrm{SHR}^{6}(x) \end{cases}$$

SHA-512の前処理

1. 入力メッセージMについて、メッセージ長が1024ビットの倍数になるように初期パディングされたメッセージ

   $$M||1||0^k||l$$

   を計算する。ただし、lはMのメッセージ長のバイナリ表現(128ビット)、kは$l+1+k \equiv 896 \pmod{1024}$を満たす正の最小値である。

2. 初期パディングされたメッセージはN個の1024ビット単位のブロック$\{M^{(i)}\}_{i=1}^{N}$に分割される。ただし、各々の$M^{(i)}$は、16個の64ビット長のワード

   $$M^{(i)} = M_0^{(i)}||M_1^{(i)}||\cdots||M_{15}^{(i)}$$

   からなる。

3. 初期値として

$$
\begin{aligned}
H_0^{(0)} &= \texttt{6a09e667f3bcc908} \\
H_1^{(0)} &= \texttt{bb67ae8584caa73b} \\
H_2^{(0)} &= \texttt{3c6ef372fe94f82b} \\
H_3^{(0)} &= \texttt{a54ff53a5f1d36f1} \\
H_4^{(0)} &= \texttt{510e527fade682d1} \\
H_5^{(0)} &= \texttt{9b05688c2b3e6c1f} \\
H_6^{(0)} &= \texttt{1f83d9abfb41bd6b} \\
H_7^{(0)} &= \texttt{5be0cd19137e2179}
\end{aligned}
$$

を設定する。

SHA-512 のハッシュ値計算

N 個のメッセージブロック $M^{(1)}, \ldots, M^{(N)}$ の $M^{(i)}$ に対して、$1 \leq i \leq N$ の順に以下を実行する。

1. 次式で定義する SHA-512 メッセージスケジュール関数を用いて拡張メッセージ W_t を計算する。

$$
W_t = \begin{cases} M_t^{(i)} & 0 \leq t \leq 15 \\ \sigma_1^{512}(W_{t-2}) + W_{t-7} + \sigma_0^{512}(W_{t-15}) + W_{t-16} & 16 \leq t \leq 79 \end{cases}
$$

2. 8 個のバッファ変数を $(i-1)$ 番目のハッシュ値 $H^{(i-1)}$ で初期化する。

$$
\begin{aligned}
a_0 &= H_0^{(i-1)} \\
b_0 &= H_1^{(i-1)} \\
c_0 &= H_2^{(i-1)} \\
d_0 &= H_3^{(i-1)} \\
e_0 &= H_4^{(i-1)} \\
f_0 &= H_5^{(i-1)} \\
g_0 &= H_6^{(i-1)} \\
h_0 &= H_7^{(i-1)}
\end{aligned}
$$

3. $0 \leq t \leq 79$ に対して以下の計算を繰り返す。

$$\begin{cases} T_1 &= h_t + \Sigma_1^{512}(e_t) + \mathrm{Ch}(e_t, f_t, g_t) + K_t^{512} + W_t \\ T_2 &= \Sigma_0^{512}(a_t) + \mathrm{Maj}(a_t, b_t, c_t) \\ h_{t+1} &= g_t \\ g_{t+1} &= f_t \\ f_{t+1} &= e_t \\ e_{t+1} &= d_t + T_1 \\ d_{t+1} &= c_t \\ c_{t+1} &= b_t \\ b_{t+1} &= a_t \\ a_{t+1} &= T_1 + T_2 \end{cases}$$

ただし、K_t^{512} は 64 ビットワードの定数 (FIPS PUB 180-2 参照) である。

4. i 番目の中間ハッシュ値を

$$\begin{aligned} H_0^{(i)} &= H_0^{(i-1)} + a_{80} \\ H_1^{(i)} &= H_1^{(i-1)} + b_{80} \\ H_2^{(i)} &= H_2^{(i-1)} + c_{80} \\ H_3^{(i)} &= H_3^{(i-1)} + d_{80} \\ H_4^{(i)} &= H_4^{(i-1)} + e_{80} \\ H_5^{(i)} &= H_5^{(i-1)} + f_{80} \\ H_6^{(i)} &= H_6^{(i-1)} + g_{80} \\ H_7^{(i)} &= H_7^{(i-1)} + h_{80} \end{aligned}$$

で計算する。

上記手続き 1.〜 4. を N 回繰り返した最終的な 512 ビットの値

$$H^{(N)} = H_0^{(N)} || H_1^{(N)} || H_2^{(N)} || H_3^{(N)} || H_4^{(N)} || H_5^{(N)} || H_6^{(N)} || H_7^{(N)}$$

がメッセージ M のハッシュ値である。

3.10 SHA-384

3.10.1 概要

SHA-384 は、SHA-256, SHA-512 とともに 2000 年に米国商務省技術標準機関 NIST により提案され、2002 年に FIPS180-2 [FIPS180-2] として制定された。

SHA-384 は、ビット長が 1024 ビットの倍数になるようにパディングされた任意のメッセージを入力として 384 ビットのハッシュ値を出力する。SHA-384 は SHA-512 とほぼ同じ仕様で、初期値と出力方法のみが異なる。

SHA-384 は ISO/IEC 10118-3 [ISO/IEC10118-3] の国際規格にも採用されている。

3.10.2 技術仕様

SHA-384 は以下の 2 点を除き、SHA-512 と同じ仕様である。

1. 初期値 $H^{(0)}$ を以下の値に設定する。

$$
\begin{aligned}
H_0^{(0)} &= \texttt{cbbb9d5dc1059ed8} \\
H_1^{(0)} &= \texttt{629a292a367cd507} \\
H_2^{(0)} &= \texttt{9159015a3070dd17} \\
H_3^{(0)} &= \texttt{152fecd8f70e5939} \\
H_4^{(0)} &= \texttt{67332667ffc00b31} \\
H_5^{(0)} &= \texttt{8eb44a8768581511} \\
H_6^{(0)} &= \texttt{db0c2e0d64f98fa7} \\
H_7^{(0)} &= \texttt{47b5481dbefa4fa4}
\end{aligned}
$$

2. 圧縮関数を N 回繰り返した最終的な 512 ビットの値 $H^{(N)}$ の左 384 ビットの値

$$H_0^{(N)}||H_1^{(N)}||H_2^{(N)}||H_3^{(N)}||H_4^{(N)}||H_5^{(N)}$$

を SHA-384 でのメッセージ M のハッシュ値とする。

第4章　安全性解析

4.1　ハッシュ関数の安全性

ハッシュ関数の安全性は、2.4 章に示した汎用攻撃に対する耐性と個々のアルゴリズムの解析結果の両面から評価することができる。

4.1.1　汎用攻撃に対する耐性指標

2.4 章に示したように、ハッシュ関数の汎用攻撃 (衝突攻撃、原像探索攻撃、別原像探索攻撃) に対する攻撃計算量の上限はハッシュ長 n にのみ依存する。汎用攻撃に対する耐性指標を表 4.1 に示すように定義する。

表 4.1: 汎用攻撃に対する耐性指標

ハッシュ長	衝突攻撃耐性	原像探索攻撃耐性	指標
$\leq$ 128 bit	$\leq 2^{64}$	$\leq 2^{128}$	C
$\leq$ 160 bit	$\leq 2^{80}$	$\leq 2^{160}$	B
$\leq$ 224 bit	$\leq 2^{112}$	$\leq 2^{224}$	A
$256 \leq n \leq 512$ bit	$\leq 2^{n/2}$	$\leq 2^{n}$	AA

電子政府推奨暗号リスト [CRYPTREC03] では、ハッシュ長が 256 ビット以上のハッシュ関数 (指標 AA) を推奨している。

4.1.2　ハッシュ関数解析による脆弱性指標

ハッシュ関数のアルゴリズムを解析し、衝突攻撃、原像探索攻撃、別原像探索攻撃に対する実際の攻撃計算量を評価した結果、表 4.1 に示す計算量の上限値より少ない計算量での攻撃が見つかった場合に、そのハッシュ関数アルゴリズムは「破れた」または「脆弱性が見つかった」という。

但し、その解析結果が、繰り返し型ハッシュ関数における「圧縮関数」レベルでの評価なのか、「ハッシュアルゴリズム全体」に対する評価なのかを区別することが重要である。

2.3.1 章で述べたように、繰り返し型ハッシュ関数 H では、圧縮関数 f を繰り返し適用することで長いメッセージのハッシュ値を計算する (図 4.1 参照)。

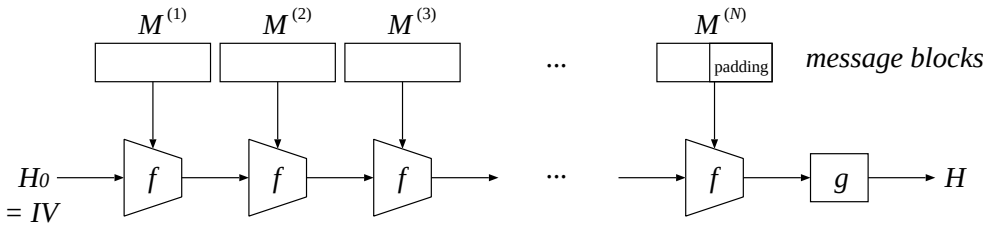

図 4.1: 繰り返し型ハッシュ関数 (再掲)

圧縮関数の衝突　ある初期ベクトル (IV) に対し、圧縮関数の出力が一致するような (1 ブロックの) 入力メッセージのペアを発見できた場合、すなわち

$$f(IV, X) = f(IV, X')$$

なる入力メッセージペア (X, X') を発見した場合、「圧縮関数の衝突が発見された」という。

圧縮関数の擬似衝突　初期ベクトル IV を任意に選べる条件下で、異なる初期ベクトル $IV'(\neq IV)$ に対し、

$$f(IV, X) = f(IV', X')$$

なる入力メッセージペア (X, X') を発見した場合、「圧縮関数の擬似衝突が発見された」という。

短縮／変形版圧縮関数の衝突　3 章で示したように、各ハッシュ関数の圧縮関数は複数のステップの繰り返しからなっており、繰返し回数 (段数) を削減したり、メッセージスケジュール関数を省略した圧縮関数の衝突解析を行っている場合もあるので注意が必要である。すなわち、圧縮関数 f を変形した関数 f' について

$$f'(IV, X) = f'(IV, X')$$

なる入力メッセージペア (X, X') を発見した場合、本報告書では「短縮／変形版圧縮関数に衝突が発見された」とする。

圧縮関数の衝突とハッシュ関数の衝突の関係　圧縮関数に衝突が見つかった場合、ハッシュ関数の衝突につながるかどうかはハッシュ関数の仕様に定められている初期値 IV_0 に対して以下のような関係が成り立つかどうかによる。

$$\begin{aligned} H(X) &= f(f(IV_0, X), P) \\ &= f(f(IV_0, X'), P) \\ &= H(X') \end{aligned}$$

ここで、P は X 及び X' に付加されるメッセージである。

衝突攻撃による安全性への影響度 このような衝突攻撃の攻撃結果が与える安全性への影響度を表 4.2 に定義する。

表 4.2: 衝突攻撃による安全性への影響度

評価結果	影響度
衝突攻撃につながる問題点は見つかっていない	0
短縮／変形版圧縮関数に衝突が発見された	1
圧縮関数に近似衝突が発見された	2
圧縮関数に擬似衝突が発見された	3
圧縮関数に衝突が発見された	4
ハッシュ関数に近似衝突が発見された	5
ハッシュ関数に擬似衝突が発見された	6
ハッシュ関数に衝突が発見された	7

これらの「汎用攻撃に対する耐性指標」と「ハッシュ関数解析結果が与える安全性への影響度」を組み合わせることで、ハッシュ関数の安全性をより適切に表現することができる。例えば、「ハッシュ長 160 ビットのハッシュ関数の圧縮関数に、(2^{80} より少ない計算量で) 衝突が発見された」場合、

このハッシュ関数の安全性は B-4

と表わすことができる。もし、衝突攻撃につながる問題点が何も見つかっていないハッシュ関数があったとしても、そのハッシュ長が 128 ビットであれば、このハッシュ関数の安全性は C-0 であり、汎用攻撃に対する耐性の観点から、長期利用には望ましくない、と判断できる。

4.2　既知の解析結果

本章では、MD4, MD5, RIPEMD, RIPEMD-128, RIPEMD-160, SHA-1, SHA-224, SHA-256, SHA-384, SHA-512, Whirlpool の各アルゴリズムについて、これまでに知られている解析結果をまとめる。

4.2.1　MD4

MD4 の解析は、1995 年以前には den Boer と Bosselaers [DB92] による解析 (圧縮関数 3 ラウンド中の最終 2 ラウンドの衝突攻撃)、及び Vaudenay [V95] による近似衝突 (“almost-collision”[1]) しか知られていなかったが、1996 年に Dobbertin により約 2^{20} 回の MD4 圧縮関数の計算量で衝突が見つけられることが発表された [D96a, D98]。この時点で既に「MD4 については利用すべきでない [D98]」とされていたが、2004 年、Wang ら [WFLY04, WLFCY04, WY04] により、極めて少ない計算量で衝突が見つけられることが発表された。MD4 の安全性は十分でなく、衝突困難性が求められる電子署名などのアプリケーションでの利用は避けるべきと思われる。

年	発表者 及び 解析結果	安全性
1992	den Boer, Bosselaers による短縮版圧縮関数の衝突攻撃 [DB92]	C-1
1995	Vaudenay による近似衝突攻撃 [V95]	C-5
1996	Dobbertin による衝突攻撃 [D96a]	C-7
2004	Wang らによる衝突攻撃 [WFLY04, WLFCY04, WY04]	C-7

4.2.2　MD5

MD5 の解析については、1993 年に den Boer と Bosselaers [DB94] により、MD5 圧縮関数に対して擬似衝突 (pseudo-collision) が発表され、1996 年に Eurocrypt’96 rump session にて、Dobbertin により、MD5 圧縮関数に対して衝突が Pentium PC で約 10 時間の計算量 (当時) で見つけることができると発表された [D96b]。この MD5 圧縮関数の衝突は、MD5 のアルゴリズム全体では擬似衝突 (pseudo-collision) になる。すなわち、仕様とは異なる別の初期値 IV' に対して、異なるメッセージからハッシュ値の一致する衝突を見つけることができるというものである。

Dobbertin は、この攻撃を踏まえて MD5 の安全性について次のように述べている。「この攻撃で MD5 の実アプリケーションが脅威にさらされるというわけではないが、その時期は近づいたと思われる。... 将来、衝突困難性が求められるアプリケーションにおいて MD5 を実装すべきでない時期がやってくるだろう [D96c]。」

その後、2004 年に Wang ら [WFLY04, WY04] により、IBM P690 で約 1 時間で衝突が見つけられることが発表された。この攻撃は、仕様通りの初期値 IV を用いて、ハッシュ値の一致する異なる 2 つの 1024 ビットメッセージを見つけるものである。

[1] 文献 [V95] では”almost-collision” の用語が使われていたが、本報告書の「近似衝突 (near-collision)」と同義.

この攻撃を利用して、実際に X.509 公開鍵証明書の偽造が可能であることも報告されている [LWW05]。衝突困難性が求められる電子署名での MD5 の利用は控えるべきと思われる。

年	発表者 及び 解析結果	安全性
1993	den Boer, Bosselaers による圧縮関数の擬似衝突攻撃 [DB94]	C-3
1996	Dobbertin による圧縮関数の衝突攻撃 [D96b] (= ハッシュ関数の擬似衝突攻撃)	C-4 (=C-6)
2004	Wang らによる衝突攻撃 [WFLY04, WY04]	C-7

4.2.3 RIPEMD

RIPEMD の解析については、1995 年に Dobbertin による、3 ラウンド中の最初の 2 ラウンド及び最終 2 ラウンドに対する衝突攻撃が知られていたが [D97]、2004 年に Wang ら [WFLY04, WLFCY04, WY04] により、RIPEMD の完全な仕様に対して衝突が見つけられることが発表された。

衝突困難性が求められる電子署名などでの RIPEMD の利用は控えるべきと思われる。

年	発表者 及び 解析結果	安全性
1995	Dobbertin による短縮版圧縮関数の衝突攻撃 [D97]	C-1
2004	Wang らによる衝突攻撃 [WFLY04, WLFCY04, WY04]	C-7

4.2.4 RIPEMD-128

現在のところ、問題点は見つかっていない。

但し、ハッシュ長が 128 ビットであるため、衝突攻撃による攻撃計算量は高々2^{64}であり、長い将来に渡って利用されるアプリケーションには適さない。

年	発表者 及び 解析結果	安全性
2005	現在のところ、問題点は見つかっていない。	C-0

4.2.5 RIPEMD-160

現在のところ、問題点は見つかっておらず、安全性上、問題はないと思われる。

但し、ハッシュ長が 160 ビットであるため、衝突攻撃による攻撃計算量は高々2^{80}であり、将来に渡って安全であるとは保証できない。

年	発表者 及び 解析結果	安全性
2005	現在のところ、問題点は見つかっていない。	B-0

4.2.6　SHA (SHA-0)

SHA (SHA-0) に対する解析については、1995 年に Chabaud と Joux により、SHA-0 の圧縮関数の衝突を 2^{61} の計算量で見つけられることが示されたのが Birthday attack より少ない計算量での最初の攻撃結果である [CJ98]。その後、2004 年に Biham と Chen により、SHA-0 の近似衝突 (near-collision) [2] や、65 段に短縮した SHA-0 の衝突などが発見された [BC04a]。さらに、2004 年 8 月に Joux らにより、SHA-0 の衝突が発見された (攻撃計算量は約 2^{51})[JCLJ04]。Biham らはその後の解析をふまえ、「衝突が脅威となるアプリケーションで SHA-0 を利用すべきでない [BC04a, updated version]」と述べている。

また、2004 年に発表された Wang らのレポート [WFLY04, WY04] には、SHA-0 の衝突が約 2^{40} 回の SHA-0 の圧縮関数の計算量で見つけられるとの記述もある。衝突困難性が求められる電子署名などのアプリケーションでは、SHA(SHA-0) の利用は控えるべきと思われる。

年	発表者 及び 解析結果	安全性
1995	Chabaud, Joux による圧縮関数の衝突攻撃 [CJ98]	B-4
2004	Biham, Chen による近似衝突攻撃 [BC04a]	B-5
2004	Joux らによる衝突攻撃 [JCLJ04]	B-7
2004	Wang らによる衝突攻撃 [WFLY04, WY04]	B-7

4.2.7　SHA-1

SHA-1 に対する解析については、2004 年まで Birthday attack より少ない計算量での攻撃は知られていなかったが、2004 年に Crypto 2004 rump session にて、Biham と Chen により、36 段に短縮した SHA-1 の圧縮関数の衝突、45 段に短縮した SHA-1 の圧縮関数の近似衝突 (near-collision) などが Birthday attack より少ない計算量で見つけられることが発表された [BC04b]。

2005 年 2 月に Rijmen と Oswald[RO04] により、SHA-1 のメッセージ拡大をうまく扱えるように Chabaud と Joux の攻撃 [CJ98] を改良し、53 段に短縮した SHA-1 の圧縮関数の衝突を発見したことが示され、さらに Wang らにより SHA-1 の衝突が 2^{69} 回の SHA-1 の計算より少ない計算量で発見できることが報告された [WYY05]。この技術詳細は現在、まだ公開されていないが、今後、関連技術文書の発表があった場合には、これを調査する必要があると考える。

年	発表者 及び 解析結果	安全性
2004	Biham, Chen による 36 段短縮版圧縮関数の衝突攻撃 [BC04b]	B-1
2004	Rijmen, Oswald による 53 段短縮版圧縮関数の衝突攻撃 [RO04]	B-1
2005	Wang, Yin, Yu による衝突攻撃 [WYY05]	B-7

[2] 160 ビットのハッシュ値のうち、142 ビットまでが一致する.

4.2.8 Whirlpool

2003年に白井、渋谷により、Whirlpoolのアルゴリズムで採用されているdiffusion matrixの分岐数 (branch number) が、提案者らの主張していた9ではなく、実際は8であったことが示され[SS03]、diffusion matrixが変更された[3]。しかしながらこの変更により、当初示されていた安全性解析には影響はない。

Whirlpoolは2000年に提案された後、NESSIEプロジェクトで評価されたが[K02a, K02b]、ハッシュ関数の安全性に直接関わる結果は発表されていない。今後の解析に注意していく必要がある。

年	発表者 及び 解析結果	安全性
2005	現在のところ、問題点は見つかっていない。	AA-0

4.2.9 SHA-256/224

SHA-256に対する解析は、GilbertとHandshuhによる評価[GH02a]が最初の公開文書である。この中で、SHA-256を含む全てのSHA-2ファミリーの衝突を見つけるための、最も確率の高い"differential collision pattern"の確率が2^{-66}であり、SHA-256の攻撃計算量は2^{132}となることから、SHA-256は衝突攻撃に対して耐性をもつという結論が導かれている。

しかしながらその後、Hawkesら[HPR04]により、この"differential collision pattern"の確率は、算術加算をオペレータとする差分定義のもとでは、より大きな値となると主張されている。

但し、Hawkesらの解析ではメッセージスケジュール関数の解析は不十分であり、単純な解析による楽観的な攻撃計算量が導かれているにすぎない[HPR04]。よって、SHA-224及びSHA-256の衝突攻撃に対する安全性を評価するにはより詳細なメッセージスケジュール関数の解析が必要であり、引続き、今後の動向に注意していく必要がある。

年	発表者 及び 解析結果	安全性
2004	Hawkesらによる圧縮関数の解析[HPR04]	A-0 (SHA-224) AA-0 (SHA-256)

4.2.10 SHA-384/512

SHA-384, SHA-512に対する解析は、GilbertとHandshuhによる評価[GH02b]が最初の公開文書である。この中で、SHA-384/512を含む全てのSHA-2ファミリーの衝突を見つけるための最も確率の高い"differential collision pattern"の確率が2^{-66}

[3]変更後の仕様がISO/IEC 10118-3に採用されている。

であり、SHA-384/512 の攻撃計算量は 2^{264} となることから、SHA-384/512 は衝突攻撃に対して耐性をもつという結論が導かれている。

しかしながらその後、Hawkes ら [HPR04] により、この”differential collision pattern” の確率は、算術加算をオペレータとする差分定義のもとでは、より大きな値となると主張されている。

但し、Hawkes らの解析ではメッセージスケジュール関数の解析は不十分であり、単純な解析による楽観的な攻撃計算量が導かれているにすぎない [HPR04]。よって、SHA-384 及び SHA-512 の衝突攻撃に対する安全性を評価するにはより詳細なメッセージスケジュール関数の解析が必要であり、引続き、今後の動向に注意していく必要がある。

年	発表者 及び 解析結果	安全性
2004	Hawkes らによる圧縮関数の解析 [HPR04]	AA-0

4.2.11　まとめ

以上の章で述べた、個別のアルゴリズムに対する既知の解析結果をもとに、汎用攻撃に対する耐性と、解析結果が与える安全性への影響度の両面から見た、現時点でのハッシュ関数の安全性を表 4.3 及び図 4.2 に示す。

表 4.3: 既知の解析結果にもとづくハッシュ関数の安全性

ハッシュ関数	安全性
MD4	C-7
MD5	C-7
RIPEMD	C-7
RIPEMD-128	C-0
RIPEMD-160	B-0
SHA(SHA-0)	B-7
SHA-1	B-7
SHA-224	A-0
SHA-256	AA-0
SHA-384	AA-0
SHA-512	AA-0
Whirlpool	AA-0

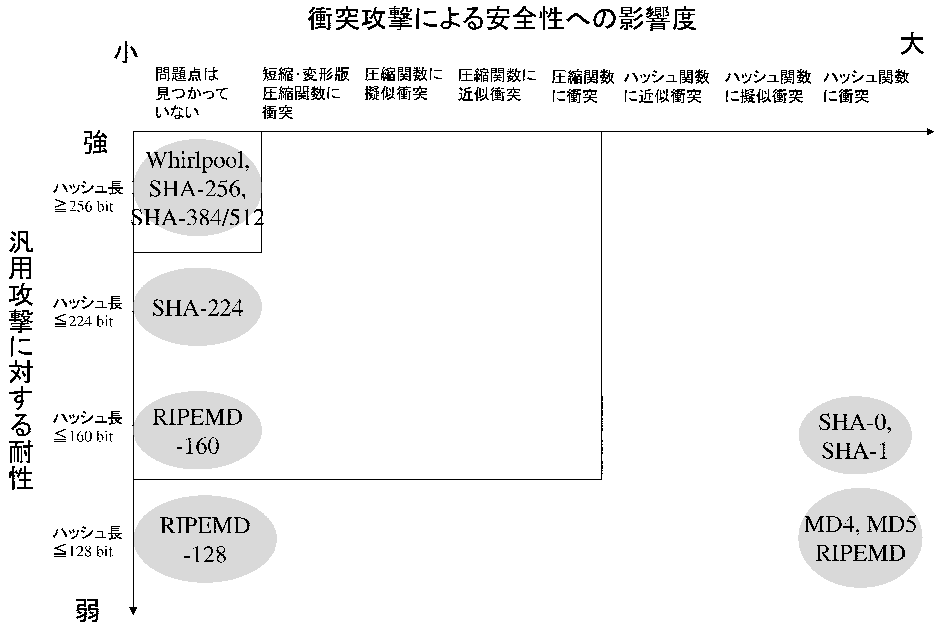

図 4.2: 既知の解析結果にもとづくハッシュ関数の安全性

第5章　ソフトウェア実装性能

表5.1及び図5.1に主な専用ハッシュ関数のソフトウェア実装性能を示す。実装性能は、入力メッセージのバイト当たりで必要なサイクル数で示している。すなわち、数値が小さいほど高速であることを意味する。

表 5.1: ソフトウェア実装性能 (cycles/byte)

アルゴリズム	PIII/Win98 (系列１)	PIII/Linux (系列２)	PIII/Win00 (系列３)	P4/Linux (系列４)
MD4	–	4.7	4.5	6.4
MD5	3.66	7.2	6.8	9.4
RIPEMD-128	6.64	–	–	–
RIPEMD-160	11.34	18	16	26
SHA (SHA-0)	–	15	12	23
SHA-1	8.30	15	13	25
SHA-256	20.59	39	39	40
SHA-384	–	83	74	122
SHA-512	40.18	83	74	122
Whirlpool	36.52	46	73	60

PIII/Win98: Pentium III (800MHz, 256MB RAM), Windows 98, Visual C++, MASM 6.15. 文献 [NM03] に示されている様々な実装方法のうち、最速のものを引用.

PIII/Linux: Pentium III (450MHz, 256MB RAM), Linux 2.4.17, gcc 3.1.1 など. 測定環境は何種類かあるが、サイクル数はほぼ同一 [NESSIE03].

PIII/Win00: Pentium III (850MHz, 256MB RAM), Windows 2000, gcc 2.95.3 など. 測定環境は何種類かあるが、サイクル数はほぼ同一 [NESSIE03].

P4/Linux: Pentium 4 (1.8GHz), Linux 2.4.0, gcc 2.95.2 など. 測定環境は何種類かあるが、サイクル数はほぼ同一 [NESSIE03].

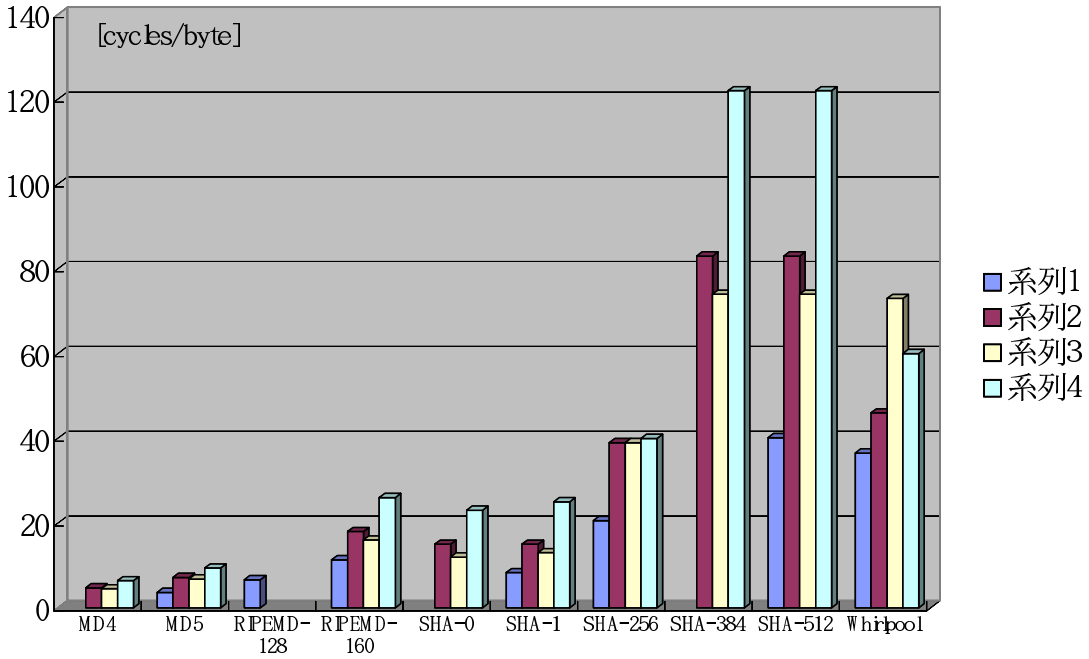

図 5.1: ソフトウェア実装性能 (cycles/byte)

第6章　まとめ

本報告書では、暗号学的ハッシュ関数、中でも特に広く利用されている専用ハッシュ関数について、その技術的特徴と採用・標準化動向について調査し、最近の解析結果を踏まえた安全性評価をまとめた。

依然として広く使われていると思われる MD4, MD5 の他、ISO/IEC 10118-3 や FIPS 180-2 として標準化されている RIPEMD-128, RIPEMD-160, SHA-1, SHA-224/256, SHA-384/512, Whirlpool についてその安全性を表 6.1, 表 6.2 にまとめた.

表 6.1: ハッシュ関数に関する安全性のまとめ (1/2)

名称	ハッシュ長 (bit)	安全性	標準
MD4	128	C-7	RFC 1320
		極めて容易に衝突を発見する方法が存在する。衝突困難性が求められる応用での利用は避けるべき。	
MD5	128	C-7	RFC 1321
		容易に衝突を発見する方法が存在する。衝突困難性が求められる応用での利用は控えていくべき。	
RIPEMD-128	128	C-0	ISO/IEC 10118-3
		現在のところ問題点は見つかっていないが、ハッシュ長が 128 ビットであるため、長い将来に渡る応用には不適。	
RIPEMD-160	160	B-0	電子政府推奨暗号 注) ISO/IEC 10118-3
		現在のところ問題点は見つかっていないが、ハッシュ長が 160 ビットであるため、将来に渡る安全性は保証できない。	

注) RIPEMD-160 については、電子政府推奨暗号リストにおいて「新たな電子政府用システムを構築する場合、より長いハッシュ値のものが使用できるのであれば、256 ビット以上のハッシュ関数を選択することが望ましい。ただし、公開鍵暗号での仕様上、利用すべきハッシュ関数が指定されている場合には、この限りではない。」という注釈がついている。

表 6.2: ハッシュ関数に関する安全性のまとめ (2/2)

名称	ハッシュ長 (bit)	安全性	標準
SHA-1	160	B-7 2^{69} 回の SHA-1 の計算より少ない計算量で衝突が発見できることが報告された。技術詳細は未公開であり、調査が必要。	電子政府推奨暗号 注) FIPS 180-2 ISO/IEC 10118-3
SHA-224/256	224/256	A-0/AA-0 現在のところ致命的な問題点は見つかっていないが、今後の動向に注意が必要。	電子政府推奨暗号 FIPS 180-2 ISO/IEC 10118-3
SHA-384/512	384/512	AA-0 現在のところ致命的な問題点は見つかっていないが、今後の動向に注意が必要。	電子政府推奨暗号 FIPS 180-2 ISO/IEC 10118-3
Whirlpool	512	AA-0 現在のところ問題点は見つかっていないが、提案者以外による安全性解析は発表されていないため、今後の動向に注意が必要。	ISO/IEC10118-3

注) SHA-1 については、電子政府推奨暗号リストにおいて「新たな電子政府用システムを構築する場合、より長いハッシュ値のものが使用できるのであれば、256 ビット以上のハッシュ関数を選択することが望ましい。ただし、公開鍵暗号での仕様上、利用すべきハッシュ関数が指定されている場合には、この限りではない。」という注釈がついている。

参考文献

[BC04a] E. Biham and R. Chen, “Near-collisions of SHA-0,” Advances in Cryptology — CRYPTO 2004, Lecture Notes in Computer Science Vol. 3152, Springer-Verlag, 2004, pp. 290–305. Updated version available at http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-get.cgi/2004/CS/CS-2004-09.ps.gz

[BC04b] E. Biham and R. Chen, “New Results on SHA-0 and SHA-1,” Short Talk Presented at CRYPTO 2004 Rump Session, 2004.

[BCS04] J. Black, M. Cochran, and T. Shrimpton, “On the Impossibility of Highly Efficient Blockcipher-based Hash-functions,” Cryptology ePrint Archive, Report 2004/062, http://eprint.iacr.org/2004/062.

[BR00] P. S. L. M. Barreto and V. Rijmen, “The Whirlpool Hashing Function,” First Open NESSIE Workshop, November, 2000, (revised May 24, 2003).

[C03a] 独立行政法人 情報処理推進機構, 通信・放送機構, ブロック暗号を使った秘匿、メッセージ認証、及び認証暗号を目的とした利用モードの技術調査報告, 2003. http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/documents/mode_wg040607_000.pdf, http://cryptrec.nict.go.jp/PDF/wat_rep040427/mode_wg040607.pdf

[CJ98] F. Chabaud and A. Joux, “Differential Collisions in SHA-0,” Advances in Cryptology — CRYPTO’98, Lecture Notes in Computer Science Vol.1462, Springer-Verlag, 1998, pp. 56–71.

[CRYPTREC03] 総務省, 経済産業省, 電子政府推奨暗号リスト平成 15 年 2 月 20 日. http://www.soumu.go.jp/joho_tsusin/security/pdf/cryptrec_01.pdf

[D89] I. B. Damgård, “A Design Principle for Hash Functions,” Advances in Cryptology — CRYPTO’89, Lecture Notes in Computer Science Vol. 435, Springer-Verlag, 1990, pp. 416–427.

[D96a] H. Dobbertin, “Cryptanalysis of MD4,” Fast Software Encryption — FSE’96, Lecture Notes in Computer Science Vol. 1039, Springer-Verlag, 1996, pp. 53–69.

[D96b] H. Dobbertin, “Cryptanalysis of MD5 Compress,” Rump Session Talk at EUROCRYPT’96, 1996.

[D96c] H. Dobbertin, “The Status of MD5 after a Recent Attack,” CryptoBytes Vol. 2, No. 2, 1996, pp. 1–6.

[D97] H. Dobbertin, “RIPEMD with Two-round Compress Function is Not Collision-free,” Journal of Cryptology Vol. 10, No. 1, 1997, pp. 51–70.

[D98] H. Dobbertin, “Cryptanalysis of MD4,” Journal of Cryptology Vol. 11, No. 4, 1998, pp. 253–271.

[DB92] 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 Vol. 576, Springer-Verlag, 1992, pp. 194–203.

[DB94] B. den Boer and A. Bosselaers, “Collisions for the Compression Function of MD5,” Advances in Cryptology — EUROCRYPT’93, Lecture Notes in Computer Science Vol. 773, Springer-Verlag, 1994, pp. 293–304.

[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, Springer-Verlag, 1996, pp. 71–82.

[FIPS180] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180, Secure Hash Standard, May 11, 1993.

[FIPS180-1] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-1, Secure Hash Standard, (supersedes FIPS 180) April 17, 1995.

[FIPS180-2] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-2, Secure Hash Standard, (supersedes FIPS 180-1) August 1, 2002.

[FIPS180-2a] National Institute of Standards and Technology, Federal Information Processing Standards Publication 180-2 with Change Notice to Include SHA-224, Secure Hash Standard, February 25, 2004.

[GH02a] H. Gilbert and H. Handschuh, "Evaluation Report, Security Level of Cryptography — SHA-256," 2002. Available at `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1045_IPA-SHA256.pdf`.

[GH02b] H. Gilbert and H. Handschuh, "Evaluation Report, Security Level of Cryptography — SHA-384 and SHA-512," 2002. Available at `http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1046_SHA_384_512.pdf`.

[GH03] H. Handschuh and H. Gilbert, "Security Analysis of SHA-256 and Sisters," Selected Areas in Cryptography — SAC 2003, Lecture Notes in Computer Science Vol. 3006, Springer-Verlag, pp. 175–193, 2004.

[H04] S. Hirose, "Provably Secure Double-block-length Hash Functions in a Black-box Model," ICISC 2004 Pre-proceedings, pp. 485–497, 2004.

[HH05] S. Hirose and M. Hattori, "A Note on Security of Double-block-length Hash Functions," The 2005 Symposium on Cryptography and Information Security (SCIS 2005) Proceedings, pp. 559–564, 2005.

[HPR04] P. Hawkes, M. Paddon, and G. G. Rose, "On Corrective Patterns for the SHA-2 Family," Cryptology ePrint Archive, Report 2004/207, `http://eprint.iacr.org/2004/207`.

[ISO/IEC10118-1] ISO/IEC 10118-1: 2000, Hash-functions – Part 1: General (2nd edition).

[ISO/IEC10118-2] ISO/IEC 10118-2: 2000, Hash-functions – Part 2: Hash-functions Using an n-bit Block Cipher Algorithm (2nd edition).

[ISO/IEC10118-3] ISO/IEC 10118-3: 2004, Hash-functions – Part 3: Dedicated Hash-functions (3rd edition).

[ISO/IEC10118-4] ISO/IEC 10118-4: 1998, Hash-functions – Part 4: Hash-functions Using Modular Arithmetic.

[JCLJ04] A. Joux, P. Carribault, C. Lemuet, and W. Jalby, “Collision in SHA-0,” Posted to `sci.crypt` NNTP News Group, August 12, 2004.

[K02a] L. R. Knudsen, “Non-random properties of reduced-round Whirlpool,” public report `NES/DOC/UIB/WP5/016`, NESSIE, June 2002.

[K02b] L. R. Knudsen, “Quadratic relations in Khazad and Whirlpool,” public report `NES/DOC/UIB/WP5/017`, NESSIE, June 2002.

[LWW05] A. Lenstra, X. Wang and B. de Weger, “Colliding X.509 Certificates,” Cryptology ePrint Archive, Report 2005/067, Available at `http://eprint.iacr.org/2005/067`, March 1, 2005.

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

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

[NESSIE03] NESSIE, “Performance of Optimized Implementations of the NESSIE Primitives, version 2.0,” February 20, 2003. Available at `https://www.cosic.esat.kuleuven.ac.be/nessie/deliverables/D21-v2.pdf`.

[NM03] J. Nakajima and M. Matsui, “Performance Analysis and Parallel Implementation of Dedicated Hash Functions on Pentium III,” IEICE Transactions on Fundamentals Vol. E86-A, No. 1, January, 2003, pp. 54–63.

[R90] R. L. Rivest, “The MD4 Message Digest Algorithm,” Advances in Cryptology — Crypto’90, Lecture Notes in Computer Science Vol. 537, Springer-Verlag, 1991, pp. 303–311.

[R92a] R. L. Rivest, “The MD4 Message-digest Algorithm,” Request for comments (RFC) 1320, IETF, 1992.

[R92b] R. L. Rivest, “The MD5 Message-digest Algorithm,” Request for comments (RFC) 1321, IETF, 1992.

[RIPE92] Research and Development in Advanced Communication Technologies in Europe, “RIPE Integrity Primitives: Final Report of RACE Integriy Primitives Evaluation (R1040),” RACE, June 1992.

[RIPE95] RIPE Consortium, RIPE Integrity Primitives — Final Report of RACE Integriy Primitives Evaluation (R1040), Lecture Notes in Computer Science Vol. 1007, Springer-Verlag, 1995.

[RO04] V. Rijmen and E. Oswald, "Update on SHA-1," Topics in Cryptology — CT-RSA 2005, Lecture Notes in Computer Science Vol. 3376, Springer-Verlag, 2005, pp. 58–71. Updated version is available at `http://eprint.iacr.org/2005/010`, January 14, 2005.

[SS03] T. Shirai and K. Shibutani, "On the Diffusion Matrix Employed in the Whirlpool Hashing Function," March 11, 2003. Available at `https://www.cosic.esat.kuleuven.ac.be/nessie/reports/phase2/whirlpool-20030311.pdf`.

[V95] S. Vaudenay, "On the Need for Multipermutations: Cryptanalysis of MD4 and SAFER," Fast Software Encryption — FSE'95, Lecture Notes in Computer Science Vol. 1008, Springer-Verlag, 1995, pp. 286–297.

[WFLY04] X. Wang, D. Feng, X. Lai, and H. Yu, "Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD," Cryptology ePrint Archive, Report 2004/199, `http://eprint.iacr.org/2004/199`, August 16 (revised August 17), 2005.

[WLFCY04] X. Wang, X. Lai, D. Feng, H. Chen, and H. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD," to be appeared in Advances in Cryptology, — EUROCRYPT 2005.

[WY04] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," to be appeared in Advances in Cryptology, — EUROCRYPT 2005.

[WYY05] X. Wang, Y. Yin, and H. Yu, "Collision Search Attacks on SHA1," February 13, 2005. Available at `http://www.infosec.sdu.edu.cn/sha-1/shanote.pdf`.

[ZPS92] Y. Zheng, J. Pieprzyk, and J. Seberry, "HAVAL — A One-way Hashing Algorithm with Variable Length of Output," Advances in Cryptology — Auscrypt'92, Lecture Notes in Computer Science Vol. 718, Springer-Verlag, 1993, pp. 83–104.