修士論文

改良Initial Structureを用いた中間一致攻撃

総合情報学専攻

(1130029) 小松原　航

指導教員 太田 和夫 教授

2013年3月25日

目次

第1章

はじめに

暗号学的ハッシュ関数 (以下，単にハッシュ関数と記す) は任意長データに対して固定長データであるハッシュ値を出力する関数であり，認証や改竄防止などで使われる．原像攻撃とは関数 H について出力 h が与えられた際に，$h = H(M)$ となるような入力 M(原像) を求める攻撃である．l ビット出力のハッシュ関数は原像攻撃に対して 2^l の計算量的安全性を持つ必要がある．

この安全性を破る手法として中間一致攻撃が Aoki らによって提案された [3]．また，中間一致攻撃の補助的な技術として Initial Structure(IS)[1] が Sasaki らによって提案された．Aoki らは IS を用いて，ハッシュ関数 SHA-0[4] 及び SHA-1[4] に対して原像攻撃を試みた結果，52 ステップの SHA-0(本来 80 ステップ)[5]，48 ステップの SHA-1(本来 80 ステップ)[5] の原像攻撃に成功した．本研究では Aoki らの IS を改良し，攻撃ステップ数を増やすためにワード単位で大雑把に構築された IS をビット単位で構築し直た．結果，54 ステップの SHA-0 及び 54 ステップの SHA-1 と，Aoki らの攻撃より多いステップ数の攻撃に成功した (成果 1)(表 1.1)．また，Sasaki らは，IS を用いて全ステップの MD5[6]，全ステップの 4-passHAVAL[7] への攻撃を試みた結果，MD5 への攻撃 [1] は 2^{45}words(words は 1 ワード 32 ビットが複数集まったもの)，4-passHAVAL への攻撃 [2] は 2^{64}words のメモリを用いることで攻撃に成功した．本研究では Sasaki らの用いた IS に対し，攻撃のメモリ量を少なくするために，IS 適用部分の計算を，メモリを減らすのに都合がいい別の計算に等価変換できないか検討し，そのような計算のパスを見つけた．結果 MD5 への攻撃は 2^{13}words，4-passHAVAL への攻撃は 2^{30}words と，佐々木らの攻撃よりも非常に少ないメモリ量での攻撃に成功した (成果 2)(表 1.2)．

攻撃対象		攻撃ステップ数	攻撃	計算量
SHA-0	[5]	52	擬似原像	$2^{151.2}$
			原像	$2^{156.6}$
	本研究	54	擬似原像	$2^{155.4}$
			原像	$2^{158.7}$
SHA-1	[5]	48	擬似原像	$2^{156.7}$
			原像	$2^{159.3}$
	本研究	54	擬似原像	$2^{155.4}$
			原像	$2^{158.7}$

表 1.1：成果 1：SHA-0/1 の攻撃段数増加

攻撃対象		攻撃ステップ数	攻撃	計算量	メモリ量
MD5	[1]	全ステップ	擬似原像	$2^{116.9}$	2^{45}
			原像	$2^{123.4}$	
	本研究	全ステップ	擬似原像	$2^{116.9}$	2^{13}
			原像	$2^{123.4}$	
4-passHAVAL	[2]	全ステップ	擬似原像	2^{224}	2^{64}
			原像	2^{241}	
	本研究	全ステップ	擬似原像	2^{226}	2^{30}
			原像	2^{242}	

表 1.2 : 成果 2:MD5 及び 4-passHAVAL への攻撃のメモリ量削減

第2章

準備

2.1 ハッシュ関数の仕様

本研究で攻撃対象とするハッシュ関数の構造を簡潔に述べる．詳細については SHA-0/1 は [4],MD5 は [6],4-passHAVAL は [7] を参照.

攻撃対象のハッシュ関数の構造の詳細は Markle-Damgård と呼ばれる構造を採用している.

2.1.1 Markle-Damgård 構造 (図 2.1)

任意長の入力メッセージ M を，SHA-0/1 及び MD5 は 512 ビットの整数倍となるように，4-passHAVAL は 1024 ビットの整数倍となるように冗長なビットを付け加えたのち（この操作をパディングと呼ぶ），$M = M_0 \| M_1 \| ... \| M_{N-1}$ と N 個の固定長のメッセージに分割し，圧縮関数 CF を用いて $h_n \leftarrow CF(h_{n-1}, M_{n-1})(n=1,...,N)$ を計算する．h_0 は仕様で定められた初期値であり，h_N がハッシュ関数の出力 H となる.

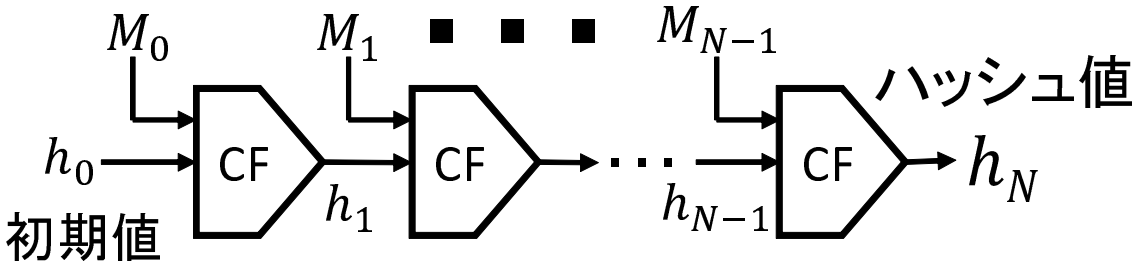

図 2.1： Markle-Damgård 構造

2.1.2 SHA-0/1 の圧縮関数

SHA-0/1 を SHA-b と表現する．つまり $b = 0$ のとき SHA-0 を，$b = 1$ のとき SHA-1 を表す．SHA-0/1 は 160 ビット出力のハッシュ関数であり，Markle-Damgård 構造を採用している.

パディングの操作は，まず入力メッセージ M の最後に 1 を付加し，M のビット長が 448(mod 512) となるまで 0 を付加する．その後入力メッセージ長の情報 64 ビットを付加する操作を行う．

以下，SHA-0/1 の圧縮関数 CF の詳細を述べる．

SHA-b の圧縮関数 CF は 80 ステップの計算を行う．計算は中間値 p_j を $p_0 \leftarrow h_i$,$p_{j+1} \leftarrow R_j(p_j, w_j)(j = 0, 1, ..., 79)$,$h_{i+1} \leftarrow h_i + p_{80}$ と更新する．

w_j は j ステップ目で使われるメッセージワードであり，入力 M_i を M_i=$m_0 \| m_1 \| ... \| m_{15}$ と 16 個の 32 ビットごとのメッセージワードに分けたとき，下記のメッセージスケジュールによって w_j が決定する．

$$\begin{cases} w_j = m_j & (0 \leq j < 16) \\ w_j = (w_{j-3} \oplus w_{j-8} \oplus w_{j-14} \oplus w_{j-16})^{\lll <<< b} & (16 \leq j < 80) \end{cases} \tag{2.1}$$

また R_j は j ステップ目のステップ関数である (図 2.2)．中間値 p_j を p_j=$a_j \| b_j \| c_j \| d_j \| e_j$ と 32 ビットごとのワードに分け，ステップ関数 R_j の計算を以下のように行う．

$$\begin{cases} a_{j+1} = a_j^{\lll 5} + F_r(b_j, c_j, d_j) + e_j + K_r + w_j \\ b_{j+1} = a_j, c_{j+1} = b_j^{\lll 30}, d_{j+1} = c_j, e_{j+1} = d_j \end{cases} \tag{2.2}$$

ここでラウンド r を r=$\lceil (j+1)/20 \rceil$ としたとき，K_r は r ラウンドで使われる定数であり，F_r は r ラウンドにおけるビット毎の論理関数である．

2.1.3 MD5 の圧縮関数

MD5 は 128 ビット出力のハッシュ関数であり，Markle-Damgård 構造を採用している．

パディングの操作は，まず入力メッセージ M の最後に 1 を付加し，M のビット長が 448(mod 512) となるまで 0 を付加する．その後入力メッセージ長の情報 64 ビットを付加する操作を行う．

以下，MD5 の圧縮関数 CF の詳細を述べる．

MD5 の圧縮関数 CF は 64 ステップの計算を行う．計算は中間値 p_j を $p_0 \leftarrow h_i,\ p_{j+1} \leftarrow R_j(p_j, m_{\pi(j)})(j = 0, 1, ..., 63),\ h_{i+1} \leftarrow h_i + p_{64}$ と更新する．

$m_{\pi(j)}$ は j ステップ目で使われるメッセージワードであり，入力 M_i を M_i=$m_0 \| m_1 \| ... \| m_{15}$ と 16 個の 32 ビットごとのメッセージワードに分けたとき，表 2.1 のメッセージスケジュールによって $\pi(j)$ が決定し，j ステップ目で使われるメッセージワード $m_{\pi(j)}$ が決定する．また R_j は j ステップ目のステップ関数である (図 2.3)．中間値 p_j を p_j=$Q_{j-3} \| Q_j \| Q_{j-1} \| Q_{j-2}$ と 32 ビットごとのワードに分け，ステップ関数 R_j の計算を以下のように行う．

$$Q_{j+1} = Q_j + (Q_{j-3} + F_r(Q_j, Q_{j-1}, Q_{j-2}) + m_{\pi(j)} + K_r)^{\lll s_j} \tag{2.3}$$

ここでラウンド r を r=$\lceil (j+1)/16 \rceil$ としたとき，K_r は r ラウンドで使われる定数であり，F_r は r ラウンドにおけるビット毎の論理関数であり，s_j はステップごとのローテーションシフトビット数である．

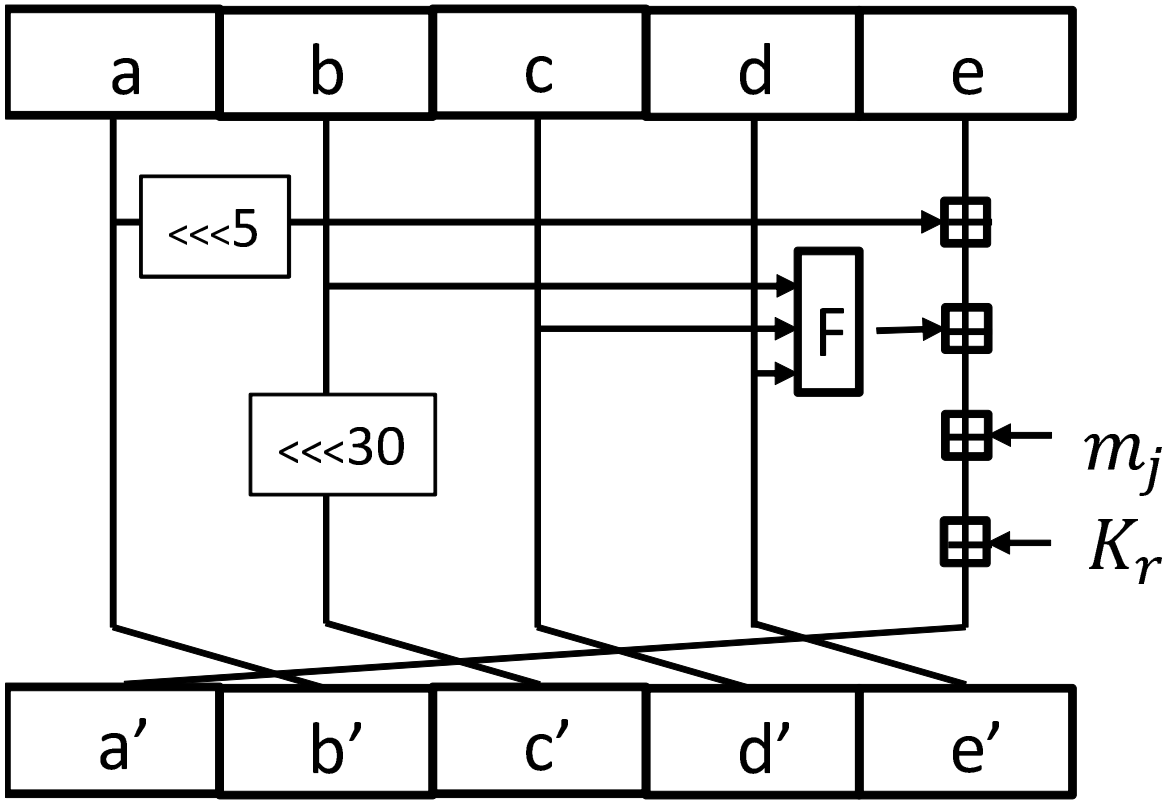

図 2.2：SHA-0/1 のステップ関数

2.1.4 4-passHAVAL の圧縮関数

4-passHAVAL は出力長が可変であり，128,160,192,224, 及び 256 ビット出力のものがある．ここでは今回攻撃した 256 ビット出力の 4-passHAVAL について述べる．

パディングの操作は，[7] を参照されたい．

4-passHAVAL の圧縮関数 CF は 128 ステップの計算を行う．計算は中間値 p_j を $p_0 \leftarrow h_i$,$p_{j+1} \leftarrow R_j(p_j, m_{\pi(j)})(j = 0, 1, ..., 127)$,$h_{i+1} \leftarrow h_i + p_{128}$ と更新する．

$m_{\pi(j)}$ は j ステップ目で使われるメッセージワードであり，入力 M_i を M_i=$m_0 \| m_1 \| ... \| m_{31}$ と 32 個の 32 ビットごとのメッセージワードに分けたとき，表 2.2 のメッセージスケジュールによって $\pi(j)$ が決定し，j ステップ目で使われるメッセージワード $m_{\pi(j)}$ が決定する．

また R_j は j ステップ目のステップ関数である (図 2.4)．中間値 p_j を p_j=$Q_{j-7} \| Q_{j-6} \| Q_{j-5} \| Q_{j-4} \| Q_{j-3} \| Q_{j-2} \| Q_{j-1} \| Q_j$ と 32 ビットごとのワードに分け，ステップ関数 R_j の計算を以下のように行う．

$$\begin{cases} T = f_r \circ \phi_r(Q_{j-6}, Q_{j-5}, Q_{j-4}, Q_{j-3}, Q_{j-2}, Q_{j-1}, Q_j) \\ Q_{j+1} = (Q_{j-7} \ggg 11) + (T \ggg 7) + m_{\pi(j)} + K_r \end{cases} \tag{2.4}$$

ここでラウンド r を r=$\lceil (j+1)/32 \rceil$ としたとき，K_r は r ラウンドで使われる定数であり，f_r は r ラウンドにおけるビット毎の論理関数であり，ϕ_r は r ラウンドにおけるワードの置換である．

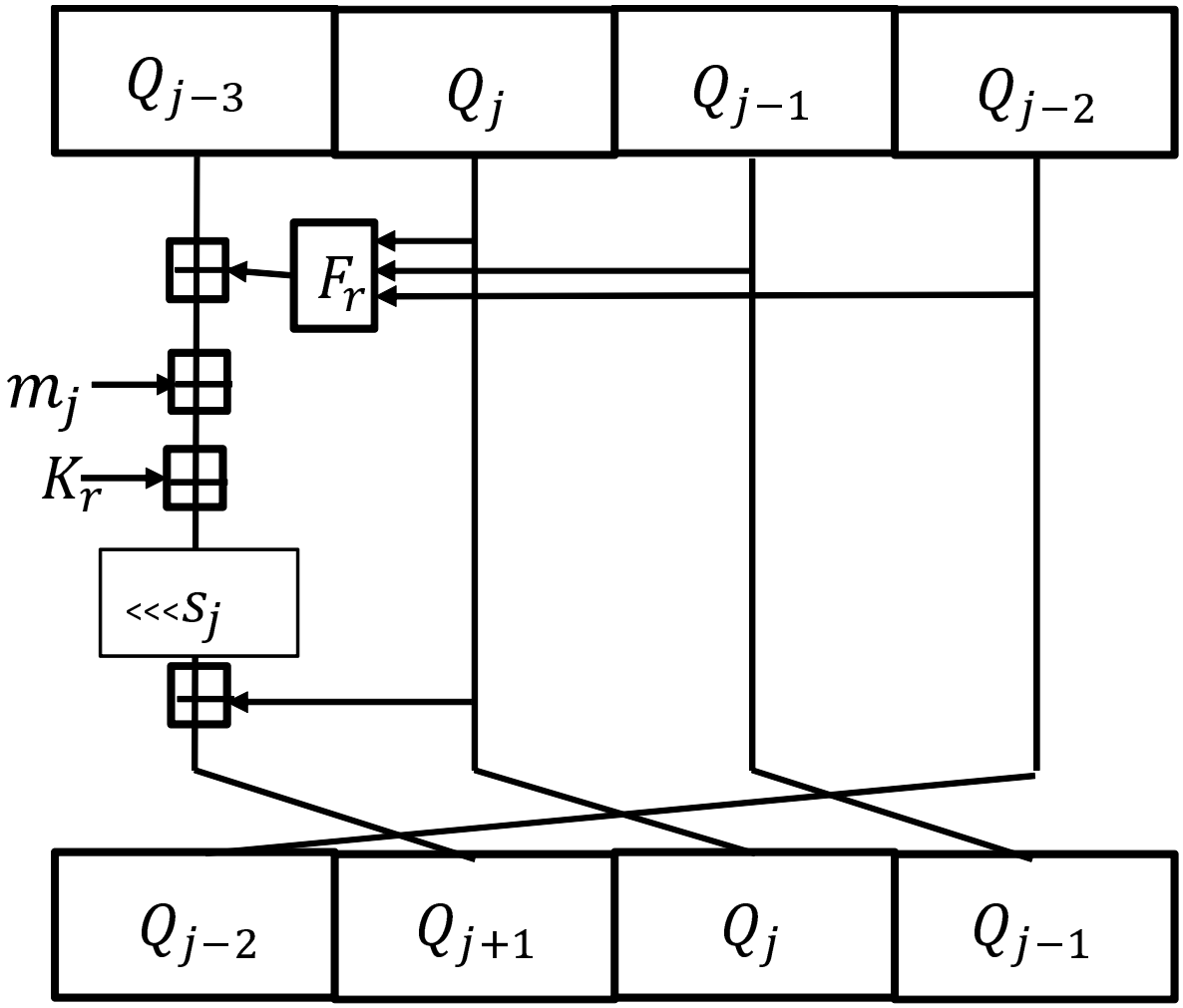

図 2.3：MD5 のステップ関数

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

表 2.1：MD5 のメッセージスケジュール

j	$\pi(j)$
0,..,31	0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
32,..,63	5 14 26 18 11 28 7 16 0 23 20 22 1 10 4 8 30 3 21 9 17 24 29 6 19 12 15 13 2 25 31 27
64,..,95	19 9 4 20 28 17 8 22 29 14 25 12 24 30 16 26 31 15 7 3 1 0 18 27 13 6 21 10 23 11 5 2
96,..,127	24 4 0 14 2 7 28 23 26 6 30 20 18 25 19 3 22 11 31 21 8 27 12 9 1 29 5 15 17 10 16 13

表 2.2： 4-passHAVAL のメッセージスケジュール

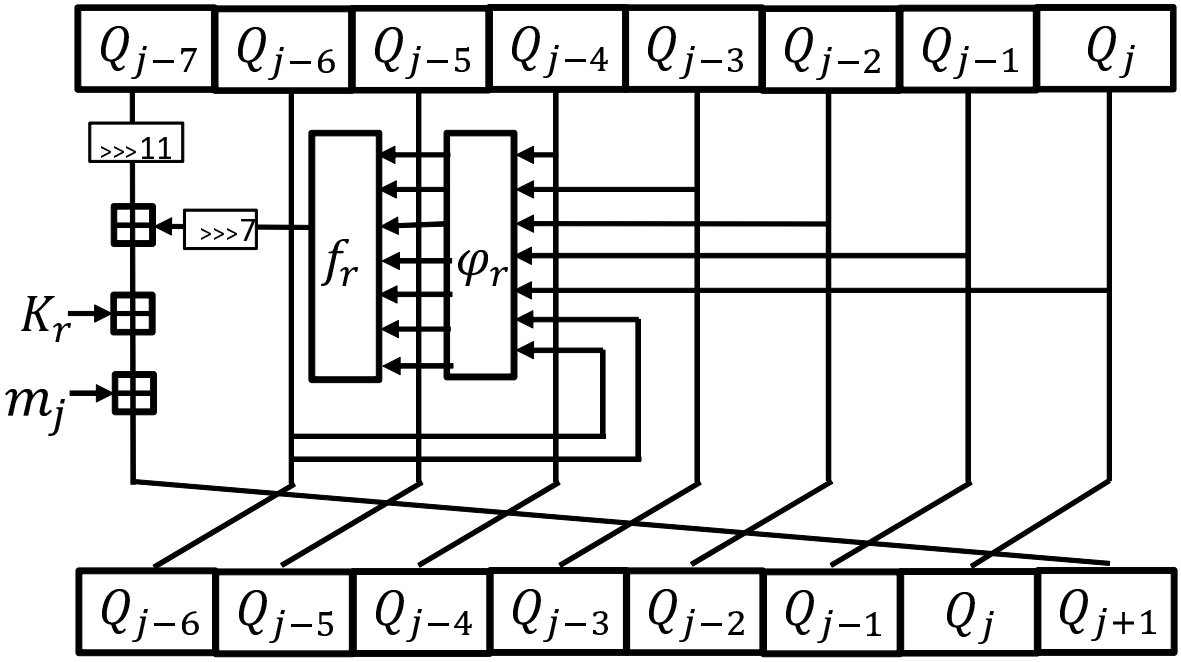

図 2.4： 4-passHAVAL のステップ関数

第3章

既存研究

3.1 擬似原像攻撃の原像攻撃への変換

擬似原像攻撃とは，圧縮関数の出力 h_i が与えられたとき，$CF(h_{i-1}, M_{i-1}) = h_i$ となるような (h_{i-1}, M_{i-1}) のペアを求める攻撃である．

x ビット出力の圧縮関数の擬似原像が計算量 2^y で求まり，かつ $y < x - 2$ で，かつ入力 M_{i-1} がハッシュ関数の最後のメッセージブロックにおけるパディングを満たすならば，ハッシュ関数の原像が計算量 $2 * 2^{(x+y)/2}$ 程度で求まる [8, Fact9.99]．

なぜなら，h_0 から h_{i-1} までの計算と，擬似原像 $(h_{i-1},\ M_{i-1})$ の計算を独立に行い，h_{i-1} が一致すれば原像を $M = M_0 \| M_1 \| ... \| M_{i-1}$ とすればよく，この手法においてそれぞれの計算のバランスをとると計算量 $2 * 2^{(x+y)/2}$ となるからである．

3.2 中間一致攻撃

中間一致攻撃は Aoki らが [3] で詳述している．以下中間一致攻撃の概要を記す．

中間一致攻撃とは，圧縮関数 CF の入力 h_{i-1} と出力 h_i が与えられたとき，$CF(h_{i-1}, M_{i-1}) = h_i$ となるような入力 M_{i-1}(圧縮関数の原像と呼ぶ) を求める場合に有効な手法である．

まず図 3.1 のように圧縮関数 CF を $CF{=}CF_B \circ CF_A$ と 2 つの関数 CF_A と CF_B の合成関数となるように分ける．このとき $M_{i-1}{=}m_A \| m_B \| m_C$ とし，m_B は CF_A の計算で使われないワードであり，m_A は CF_B の計算で使われないワードであり，m_C は CF_A 及び CF_B どちらでも使われるビット集合であるように分ける．なお，m_A,m_B のように，片方の計算と独立なワードを”中立ワード”と呼ぶ．

それぞれの中立ワードを r ビット，中間値 p_t のビット数を n ビットとすると，以下のように攻撃を行う．

手順 1　m_C をランダムな値に固定する．

手順 2　2^r 通りの値の m_A を用いて，中間値 $p_t{=}CF_A(m_A)$ を 2^r 回計算し，2^r 個の (p_t,m_A) の値をテーブル T に保存する．

手順 3　2^r 通りの値の m_B を用いて，中間値 $p_t{=}CF_B^{-1}(m_B)$ を 2^r 計算し，計算した中間値 p_t がテーブル T に含まれている p_t と一致するかどうかチェックする．もしあればそのときの

M_{i-1}=$m_A \| m_B \| m_C$ が圧縮関数の原像となる．

手順 4　手順 3 で一致しなかった場合，手順 1 に戻る．

CF_A もしくは CF_B の計算を計算量 $\frac{1}{2}$ と考える．手順 3 までを行うと，2^r 回の計算量で，2^{2r} 回の一致を確かめられる．また，手順 3 で一致が成功する確率は 2^{-n} であり，手順 3 までの計算を 2^{n-2r} 回繰り返すと，2^n 回の一致が確かめられるので，圧縮関数の原像が求まる．結局 $2^{n-r}(=(2^r)*(2^{n-2r}))$ の計算量で圧縮関数の原像が求まる．また，メモリは手順 2 で主に使われ，2^r 個の (p_t,m_A) を保存できるメモリ量が必要となる (このような場合，単に 2^rwords のメモリ量が必要と書くことにする)．

以下，中間一致攻撃で擬似原像を求めるための補助的な技術を紹介する．これらの詳細は [3] や [1] を参照．

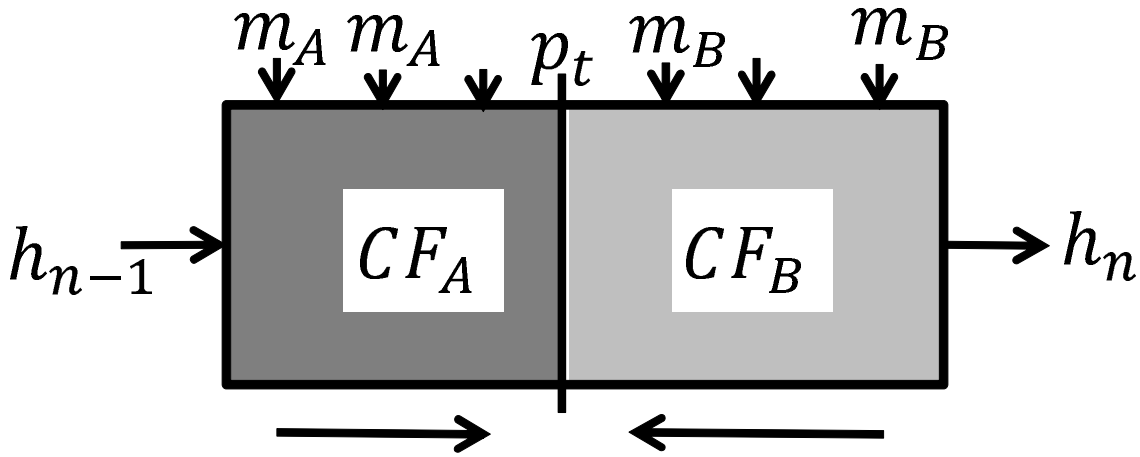

図 3.1：中間一致攻撃

3.2.1　Splice-and-Cut

ハッシュ関数は中間一致攻撃の耐性を持たせるために圧縮関数の最初と最後のステップを加算する構造が多い．Aoki と Sasaki はこれに対して Splice-and-Cut という技術を提案した [3]．

中間一致攻撃を用いて擬似原像攻撃を行うとき，圧縮関数の最初と最後のステップを繋がっているとみなす．すると図 3.2 のように入力側，出力側などを考えずに圧縮関数 CF の計算を CF_B,CF_A に分けることができるので，関数の分け方を選びやすくなる．そのため攻撃可能ステップ数が伸びる可能性が高くなる．

ただしこの技術は入力 h_{i-1} を攻撃者の任意にとれる場合にのみ有効なので，攻撃は擬似原像攻撃となる．

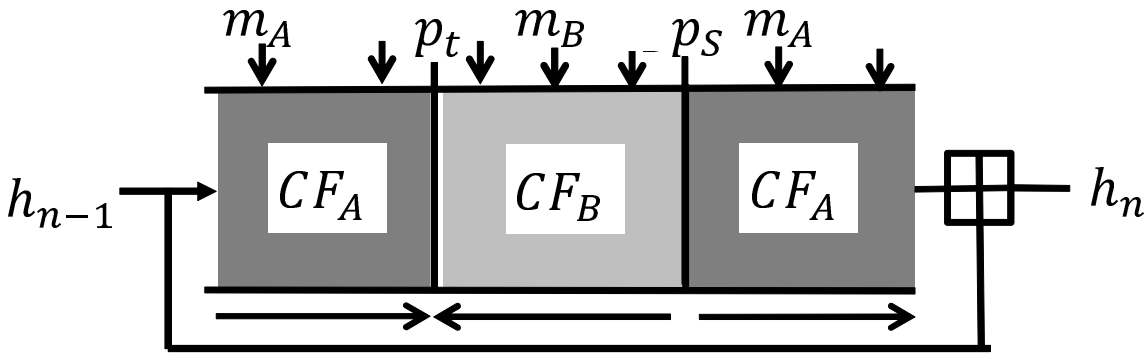

図 3.2 : Splice-and-Cut

3.2.2 Initial Structure(IS)[1]

IS は中間一致攻撃の併用技術であり，独立計算可能なステップ数を増やす技術である．図 3.3 では m_A から CF_A を計算，m_B から CF_B を計算している．このときそれぞれの計算の影響するビットが重複しないように計算することで，CF_A と CF_B を独立に計算出来る．IS を用いることで図 3.2 のような簡単な分け方に加えて，より複雑な分け方を行うことができるのでより多いステップ数の関数に攻撃できる分け方が選べるようになる．

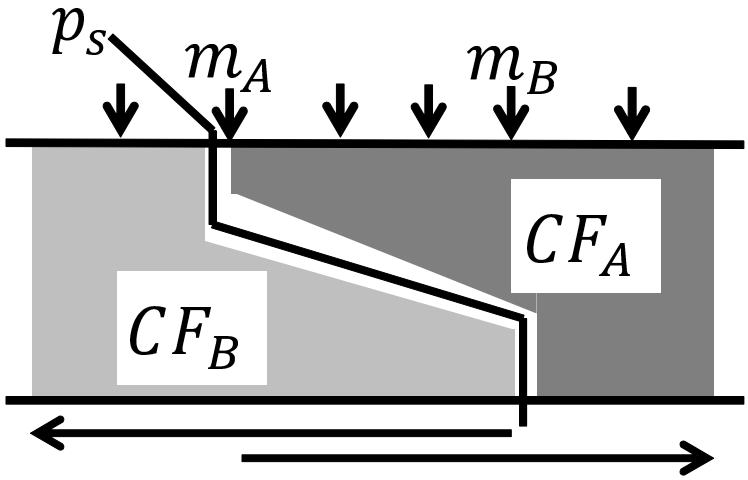

図 3.3 : Initial Structure

3.2.3 Partial Matching(PM)

PM は 2 つに分けた計算を完全には独立に行えないステップがあっても，そのステップを部分的に独立に計算することで，独立計算可能なステップを伸ばす技術である．図 3.4 のように中間一致攻撃を試みる．一致は中間値 p_t で確かめる．CF_A の計算を p_F まで行うが，p_t を計算するまでに m_B が入ってくるので p_t のビット全てを求めることはできない．しかし m_B

が入った後もいくつかのステップは部分的に m_B と独立に計算できる．そこで p_t を部分的に計算する．CF_B を p_S まで計算した後も同様に p_t を部分的に計算する．CF_A,CF_B それぞれから p_t の同じビットを求めた場合，そのビットを用いて一致を確かめることができる．

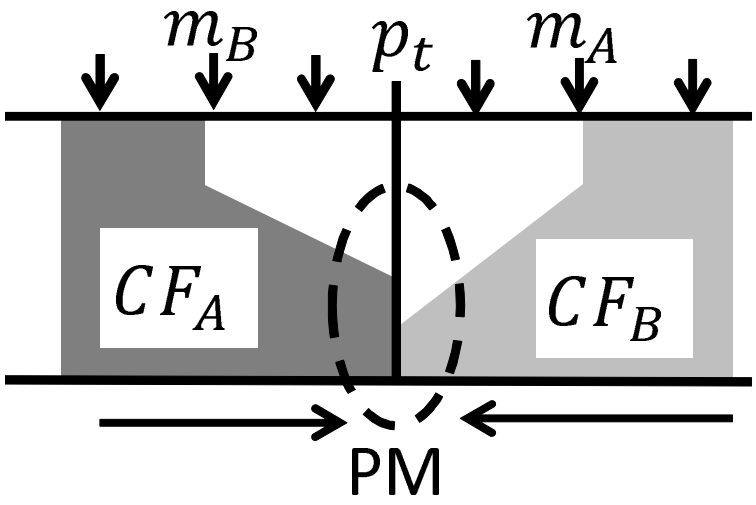

図 3.4 : Partial Matching

第4章

成果１：攻撃ステップ数の増加

Aoki らは IS を用いた中間一致攻撃を SHA-0 及び SHA-1 に対して適用し，52 ステップの SHA-0 及び 48 ステップの SHA-1 に対して攻撃可能であることを示した．本研究では Aoki らが用いた IS を改良し，IS の適用ステップ数を増やしたものを用いて，SHA-0 及び SHA-1 に対して攻撃を行い，54 ステップの SHA-0 及び 54 ステップの SHA-1 に対する攻撃に成功した．

4.1 ビット単位の IS

4.1.1 既存研究の IS：ワード単位の IS

Aoki らが用いた IS はワード (32 ビット) 単位で大雑把に構築したものであり，SHA-0 及び SHA-1 に対しては最大 4 ステップの IS が構築可能であるとした (図 4.1)．

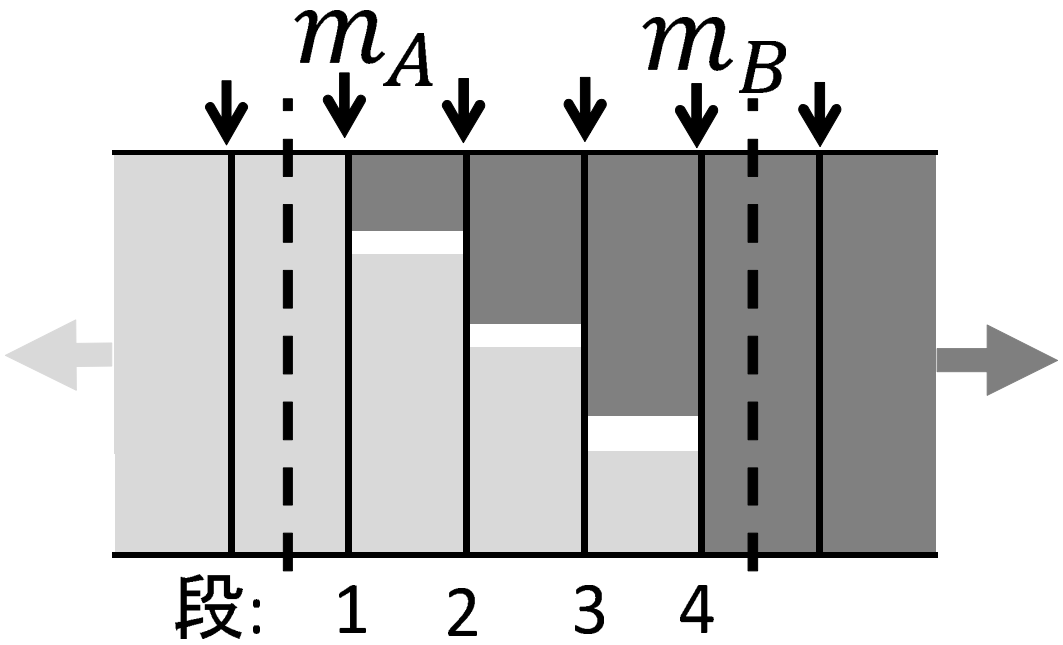

図 4.1：ワード単位で構築した IS

4.1.2 本研究の IS：ビット単位の IS

Aoki らは IS をワード単位で大雑把に構築していたが，本研究ではビット単位で詳細に構築したことで IS のステップ数を増やした．SHA-0 及び SHA-1 に対しては青木らが最大 4 ステップの IS を構築可能としたのに対し，本研究では最大 7 ステップの IS を構築可能とした．本研究の IS の概要図を図 4.2 に示す．IS のステップ数が増えたことにより独立計算可能なステップ数が増え，関数の分け方をより多く選べるため，多いステップ数のハッシュ関数に対して攻撃可能となるような分け方を選べる．

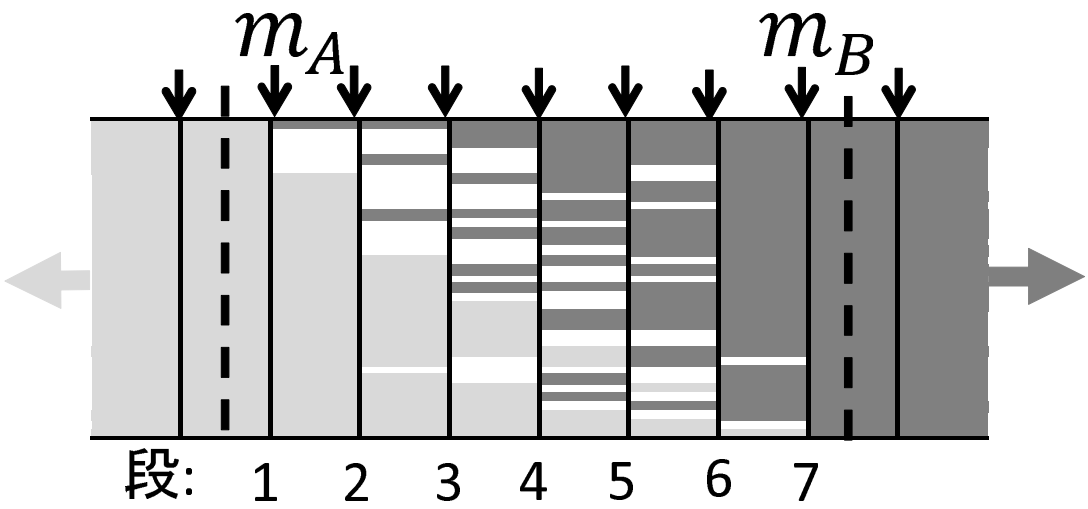

図 4.2：ビット単位で構築した IS

4.2 54 ステップ SHA-0 への攻撃

4.2.1 攻撃概要

IS において，Aoki らは中間値の 5 つのワードのうち，最低 1 つはいずれかの中立ワードと独立になるような IS を考えたが，高々4 ステップの IS しか構築できなかった．我々は中間値のビットごとに同様に独立性を考えた結果，7 ステップ以上の IS が構築可能であることを発見した．中間一致攻撃を行う場合，関数をどのように分けるかが重要である．Aoki らは [5] で SHA-0 の関数分けの方法を述べている．この方法では IS と PM の合計ステップ数によって SHA-0 への攻撃ステップ数が決まる．Aoki らは IS と PM を行うステップ数の合計が 16 から 21 のとき 52 ステップの SHA-0 に攻撃が行え，ステップ数の合計が 22 のとき 54 ステップ版の SHA-0 に対して攻撃が行えることを発見した．また，Aoki らは 2 ステップの IS と 14 ステップの PM を構築し，IS と PM の合計ステップ数を 16 として 52 ステップの SHA-0 に対して攻撃を行なった．本論文では 7 ステップの IS と 15 ステップの PM を構築し，IS と PM の合計ステップ数を 22 として 54 ステップの SHA-0 に対して攻撃を行うことができることを示す．

表 4.1：54 ステップ SHA-0 の関数分け

			m_S															
j	m_A	m_B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
2	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	↓
3	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	
4	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
5	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	
6	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
7	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	
8	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	
9	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	PM
10	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
11	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	
12	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	
13	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	
14	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	
16	1	1	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	0	↑
17	0	1	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	
18	0	1	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	
19	0	0	1	0	1	1	0	1	0	0	1	0	0	1	0	1	0	0	
20	0	0	0	1	0	1	1	0	1	0	0	1	0	0	1	0	1	0	
21	0	1	0	0	1	0	1	1	0	1	0	0	1	0	0	1	0	1	
22	0	0	1	0	1	1	0	1	1	0	0	0	0	1	0	1	1	0	CF_B
23	0	0	0	1	0	1	1	0	1	1	0	0	0	0	1	0	1	1	
24	0	1	1	0	0	0	1	1	0	1	0	0	0	0	0	0	0	1	
25	0	1	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	0	
26	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	0	

			m_S															
j	m_A	m_B	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
27	0	0	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	1	
28	0	1	1	0	1	1	1	1	0	0	0	1	0	0	0	1	0	0	
29	0	0	0	1	0	1	1	1	1	0	0	0	1	0	0	0	1	0	
30	0	0	0	0	1	0	1	1	1	1	0	0	0	1	0	0	0	1	
31	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	1	0	0	↑
32	1	1	0	1	0	1	1	0	1	1	1	0	0	0	0	1	1	0	
33	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	0	1	1	
34	0	1	1	0	1	1	0	1	1	0	0	1	1	0	0	1	0	1	IS
35	1	0	1	1	1	1	1	0	1	1	1	0	1	1	0	1	1	0	
36	0	1	0	1	1	1	1	1	0	1	1	1	0	1	1	0	1	1	
37	0	0	1	0	0	1	1	1	1	0	0	1	1	0	1	0	0	1	
38	1	1	1	1	1	0	1	1	1	1	1	0	1	1	0	0	0	0	
39	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	0	0	0	↓
40	1	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	0	0	
41	1	0	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	0	
42	0	0	0	0	0	0	1	1	1	0	1	1	1	1	1	0	1	1	
43	0	0	1	0	1	0	0	1	1	1	1	1	1	1	1	0	0	1	
44	1	0	1	1	1	1	0	0	1	1	0	1	1	1	1	1	0	0	
45	0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0	CF_A
46	0	0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	1	0	
47	1	0	0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	1	
48	0	0	1	0	1	0	1	1	1	1	1	1	0	1	1	0	0	1	
49	0	0	1	1	1	1	0	1	1	1	0	1	0	1	1	1	0	1	
50	1	0	1	1	0	1	1	0	1	1	0	0	1	0	1	0	1	0	
51	1	0	0	1	1	0	1	1	0	1	1	0	0	1	0	1	0	1	
52	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	1	1	0	
53	1	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	1	1	

4.2.2 関数分けおよび中立ワード

CF_B と独立な中立ワードを $m_A(=m_2 = m_3 = m_6 = m_{10} = m_{12} = m_{14}$(24-31 ビット)), CF_A と独立な中立ワードを $m_B(=m_2 = m_5 = m_8 = m_{12} = m_{13} = m_{14}$(15-19 ビット)) として扱う．どちらの中立ワードも m_2,m_{12},m_{14} を用いるが，中立ワードとして使うビット位置が重複していないので，これらの中立ワードは独立に動かせる．

また，表 4.1 のように関数を分けた．j ステップ目の入力ワード w_j は式 (2.1) で決定されるが，w_j がどのメッセージワード $m_S(S = 0, 1, ..., 15)$ の影響を受けるかも表 4.1 で示している．例えば表 4.1 のステップ 16 の行を見ると，m_S の数字が 0,2,8,13 の列が”1”になっており，これは 16 ステップ目の入力ワードが $w_{16} = m_0 \oplus m_2 \oplus m_8 \oplus m_{13}$ であることを示している．また，m_A が”1”のステップは m_A の影響を受けたワードが加算される．m_B についても同様である．

なお，m_{13} の 0 ビット目を 1，m_{15} の 0-8 ビットを 10 進数で 447 と固定値にしている．これは擬似原像攻撃を原像攻撃に変換する際にパディングを満たすためであり，満たす方法は [9] を参照.

4.2.3 Initial Structure

図 4.3 のように 7 ステップの IS を構築した．目標は p_{39} の値が m_B に依存しない，かつ，p_{32} の値が m_A に依存しないようにすることである．R_{32} から R_{38} まで計算したときの m_A による丸め差分と R_{38} から R_{32} まで m_B を用いて逆算したときの丸め差分のビット位置が重複しないようにできれば目標が達成できる．

図 4.3 の左側の下線つきの数字は m_A による丸め差分を表しており，右側の下線つきの数字は m_B による丸め差分を表している．また，τ_j は 32 ビットの中間値を表している．中間

値 τ_j のうち，いくつかのビットを固定値にしている．固定値にしたビットと値を表 4.2 に示す．

以下，$\mathbf{0}$=00...0,$\mathbf{1}$=1...1 とする．

加算を分ける　32 ビットの加算 $y = c+x$ について，$x = x_1 \| x_2$ とすると，$x = (x_1 \| \mathbf{0}) + (\mathbf{0} \| x_2)$ と表すことができ，$y = c + (x_1 \| \mathbf{0}) + (\mathbf{0} \| x_2)$ と x の加算を 2 つに分けることができる．図 4.3 を見ると，メッセージワードが w_j^1 や w_j^2 と表されている箇所があるが，これは w_j の加算を 2 つに分けたものである．なお，w_j^1 は m_B と独立に計算できるビットのみを含み，w_j^2 は m_A と独立に計算できるビットのみを含む．同様に論理関数 F_2 による加算も F_2^1,F_2^2 と加算を 2 つに分けた箇所があり，$a_j^{\lll 5}$ の加算も $a_j^{1\lll 5}$,$a_j^{2\lll 5}$ と 2 つに分けた箇所があり，図 4.3 では $\lll 5$ の直前に，a_j^1,a_j^2 と表記している．表 4.3 に加算をどのように分けたかを示す．

また，図 4.3 ではメッセージワードの隣に，括弧で m_A もしくは m_B と表記されているところがあるが，これはそのメッセージワードが m_A もしくは m_B の影響を受けることを示す．まず m_A による丸め差分パスを考える．

ステップ 32　$a_{33} \leftarrow \tau_{32} + w_{32}^1$ を計算する．このとき a_{33} の 24-31 ビットに m_A の影響が出る．

ステップ 33　$a_{34} \leftarrow \tau_{33} + a_{33}^{1\lll 5}$ を計算する．このとき τ_{33} の値によっては繰り上がりによって a_{34} の 5 ビット目に丸め差分が入る場合があるが，τ_{33} の 0-16 ビットを $\mathbf{0}$ にしているので繰り上がりは発生せず，a_{34} の 0-4,29-31 ビットに丸め差分が入る．

ステップ 34　$a_{35} \leftarrow \tau_{34} + F_2^1(b_{34}, c_{34}, d_{34}) + a_{34}^{1\lll 5}$ を計算する．このとき $\tau_{34} + F_2^1(b_{34}, c_{34}, d_{34}) = \mathbf{0}$(0-23 ビット) としているので，$a_{35}$ の 10 ビット目には繰り上がりは発生せず，a_{35} の 2-9,24-31 ビットに丸め差分が入る．

ステップ 35　$a_{36} \leftarrow \tau_{35} + w_{35} + K_2 + F_2^1(b_{35}, c_{35}, d_{35}) + a_{35}^{\lll 5}$ を計算する．$w_{35} = m_0 \oplus m_1 \oplus m_2 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_{10} \oplus m_{11} \oplus m_{13} \oplus m_{14} = -K_2$(0-23 ビット) とすれば，$\tau_{35} + w_{35} + K_2 = \mathbf{0}$(0-14 ビット) となり，$a_{36}$ の 15 ビット目には繰り上がりは発生せず，a_{35} の 0-14,22-31 ビットに丸め差分が入る．

ステップ 36　$a_{37} \leftarrow \tau_{36} + w_{36}^1 + K_2 + F_2^1(b_{36}, c_{36}, d_{36}) + a_{36}^{\lll 5}$ を計算する．a_{37} にどのような丸め差分が出ても IS は構築可能であるので，丸め差分を考慮しない．図 2 では丸め差分を考慮しない場合は丸め差分を”?”で示している．

ステップ 37　$a_{38} \leftarrow e_{37} + \tau_{37} + w_{37} + F_2^1(b_{37}, c_{37}, d_{37}) + K_2 + a_{37}^{\lll 5}$ を計算する．a_{38} に入る丸め差分は考慮しない．

ステップ 38　$a_{39} \leftarrow e_{38} + \tau_{38} + w_{38}^1 + F_2(b_{38}, c_{38}, d_{38}) + K_2 + a_{38}^{\lll 5}$ を計算する．a_{39} に入る丸め差分は考慮しない．

次に m_B による丸め差分パスを考える．

表 4.2： τ_j の固定値にするビット位置及び値

ステップ j	τ_j
32	0-16,22-31=**0**
33	0-16,22-31=**0**
34	0-31=**0**
35	0-14,22-31=**0** 15-21=**1**
36	0-14,21-31=**0** 15-20=**1**
37	15-19=**1**
38	15-19=**1**

ステップ 38　$e_{38} \leftarrow \tau_{38} - w_{38}^2$ を計算する．このとき τ_{38} の値によっては繰り下がりによって e_{38} の 20 ビット目に丸め差分パスが入る場合があるが，τ_{38} の 15-19 ビットを **1** にしているので繰り下がりは発生せず，e_{38} の 15-19 ビットに丸め差分が入る．

ステップ 37　$e_{37} \leftarrow \tau_{37} - F_2^2(b_{37}, c_{37}, d_{37})$ を計算する．τ_{37} の 15-19 ビットを **1** にしているので e_{37} の 20 ビット目に繰り下がりは発生せず，e_{37} の 15-19 ビットに丸め差分が入る．

ステップ 36　$e_{36} \leftarrow \tau_{36} - F_2^2(b_{36}, c_{36}, d_{36}) - w_{36}^2$ を計算する．τ_{36} の値によらず，e_{36} の 20 ビット目には繰り下がりが発生し，丸め差分が入る．これ以上の丸め差分の拡大を抑えるため e_{36} の 21 ビット目に繰り下がりが発生しないようにする．τ_{36} の 15-20 ビットを **1** にしているので e_{36} の 21 ビット目に繰り下がりは発生せず，e_{36} の 15-20 ビットに丸め差分が入る．

ステップ 35　$e_{35} \leftarrow \tau_{35} - F_2^2(b_{35}, c_{35}, d_{35})$ を計算する．τ_{35} の 15-21 ビットを **1** にしているので e_{35} の 22 ビット目に繰り下がりは発生せず，e_{35} の 15-21 ビットに丸め差分が入る．

ステップ 34　$e_{34} \leftarrow \tau_{34} - K_2 - F_2^2(b_{34}, c_{34}, d_{34}) - w_{34}^2 + a_{34}^{2\ll 5}$ を計算する．e_{34} に入る丸め差分は考慮しない．

ステップ 33　$e_{33} \leftarrow a_{34} - \tau_{33} - K_2 - F_2(b_{33}, c_{33}, d_{33}) - w_{33} + a_{33}^{2\ll 5}$ を計算する．e_{33} に入る丸め差分は考慮しない．

ステップ 32　$e_{32} \leftarrow a_{33} - \tau_{32} - K_2 - F_2(b_{32}, c_{32}, d_{32}) - w_{32}^2 + a_{32}^{2\ll 5}$ を計算する．e_{32} に入る丸め差分は考慮しない．

a_{33} や a_{34} には m_A,m_B による丸め差分がどちらも入っているが，丸め差分のビット位置は重複していない．よって m_A と m_B の丸め差分のビット位置はひとつも重複しない．これで目標は達成でき，IS が構築できた．

4.2.4 Partial Matching

15 ステップを部分的に独立に計算し，13 ビットだけ中間値の一致を確かめられるような PM を適用した．CF_A の計算で p_2 を求め，CF_B の計算で p_{17} を求めており，これらの中間値から p_6 を部分的に求めて一致を確かめる．表 4.4 に今回の Partial Matching で，どの部分を計算したかを示す．

CF_A 側から p_6 を求めていく．

a_3 の計算　$a_3 \leftarrow e_2 + K_1 + F_1(b_2, c_2, d_2) + a_2^{\lll 5} + w_2$ を計算したいが，w_2 の 15-19 ビットは本来 m_B の影響を受けているため，値がわからない．そのため a_3 は 0-14,20-31 ビットのみが部分的に計算できる．このとき 20-31 ビットが 19 ビット目からの繰り上がりの影響を受けるか受けないかわからないので，繰り上がる場合と繰り上がらない場合のどちらの a_3 の値も候補に入れて考える．このステップで候補は 2 倍になる．

a_4 の計算　a_4 を計算するとき，$a_3^{\lll 5}$ の加算によって 20-24 ビットに未知の値が入る，a_4 は 0-19,25-31 ビットのみが部分的に計算できる．このうち 0-19,27-31 ビットの値のみを用いて PM を行っていきたい．すると 27-31 ビットへの繰り上がりを次のように考えることができる：繰り上がりを考えずに 25,26 ビットを求める．求めた値が”11”でない場合は 27-31 ビットに繰り上がりは入らない．”11”のときだけ繰り上がる場合と繰り上がらない場合のどちらの値も候補に入れて考えるとすると，このステップでは $\frac{1}{4}$ の確率で候補が 2 倍になる．平均するとこのステップでは候補が $\frac{5}{4}$ 倍になる．候補の増え方におけるこの考え方を”アイデア 1”と呼ぶことにする．

a_5 の計算　a_5 を計算するとき，w_4 によって 25-31 ビットに未知の値が入り，論理関数 F_1 によって 15-19 ビットに未知の値が入る．a_5 は 0-14,20-24 ビットのみが部分的に計算できる．20-24 ビットへの繰り上がりを考えると，このステップでは候補が 2 倍になる．

a_6 の計算　a_6 を計算するとき，論理関数 F_1 によって 13-24 ビットに未知の値が入り，$a_5^{\lll 5}$ の加算によって 0-4,20-24,30-31 ビットに未知の値が入る．a_6 は 5-12,25-29 ビットのみが部分的に計算可能だが，このうち 9-12 ビットの値のみを用いて PM を行いたい．9-12 ビットへの繰り上がりをアイデア 1 を用いて考えると，このステップでは候補が平均して $\frac{17}{16}$ 倍になる．

CF_B 側から p_6 を求めていく．

e_{16}, e_{15}, e_{14} の計算　e_{16}, e_{15}, e_{14} は全て 0-23 ビットのみが部分的に計算できる．繰り下がりが発生することはないので候補は増えない．

e_{13} の計算　e_{13} を計算するとき，論理関数 F_1 によって 0-1,24-31 ビットに未知の値が入る．よって 2-23 ビットのみが部分的に計算でき，繰り下がりにより候補は 2 倍になる．

e_{12} の計算　まず $\tau_{12} \leftarrow a_{13} - K_1 - w_{12} - F_1(b_{12}, c_{12}, d_{12})$ を計算する．このとき論理関数 F_1 によって 0-1,24-31 ビットに未知の値が入り，τ_{12} は 2-23 ビットが計算可能である．このう

表 4.3：加算の分け方

	w_j		F		a_j	
ステップ j	w_j^1	w_j^2	F_2^1	F_2^2	a_j^1	a_j^2
32	24-31	0-23	–		–	
33	–		–		0-16,22-31	17-21
34	–		24-31	0-23	0-16,22-31	17-21
35	–		0-4,22-31	5-21	–	
36	0-14,20-31	15-19	0-14,20-31	15-19	–	
37	–		0-14,20-31	15-19	–	
38	0-14,20-31	15-19	–		–	

ち 7-23 ビットのみを以降の計算で使うとすると，アイデア 1 より候補は $\frac{33}{32}$ 倍になる．また，$e_{12} \leftarrow \tau_{12} - a_{12}^{\lll 5}$ を計算すると，e_{12} の 7-23 ビットを計算できる．このとき繰り下がりを考えると候補は 2 倍になる．

e_{11}, e_{10}, e_9 の計算　まず $\tau_{11} \leftarrow a_{12} - K_1 - w_{11}$ を計算する．a_{12} は 2-25 ビットの値しかわからないため，普通に計算した場合 2 回繰り下がりを考えなければならない．そこでメッセージワード $w_{11} = m_{11} = -K_1$ となるように 0-1 ビットを固定値にしておけば,$-K_1 - w_{11} =$ "00"(0-1 ビット) となり，繰り下がりを考えなくて済む．また，$e_{11} \leftarrow \tau_{11} - F_1(b_{11}, c_{11}, d_{11}) - a_{11}^{\lll 5}$ を計算すると，e_{11} は 7-23 ビットを計算できる．ただし論理関数 F_1 と $a_{11}^{\lll 5}$ によって e_{11} の 7-23 ビットは 2 回繰り下がりを考えなければならない．このとき，1 回も繰り下がりが入らない場合，1 回だけ繰り下がりが入る場合，2 回とも繰り下がりが入る場合の 3 つの候補を考える．すると候補は 3 倍になる．候補の増え方におけるこの考え方を"アイデア 2"と呼ぶことにする．e_{10} の計算も e_{11} の計算と似ており，$w_{10} = m_{10} = -K_1$(0-1 ビット) とすれば 2 回の繰り下がりを考えて，e_{10} の 7-23 ビットを計算できる．アイデア 2 を用いて考えれば，候補は 3 倍になる．e_9 も同様に，$w_9 = m_9 = -K_1$(0-1 ビット) とすれば 2 回の繰り下がりを考えて，e_9 の 9-23 ビットを計算でき，候補は 3 倍に増える．

ここまで計算を行えば，a_5=$e_{10}^{\lll 30}$，a_6=$e_9^{\lll 30}$ は簡単に計算でき，a_6 の 9-12 ビットと a_5 の 11-14,20-24 ビットで一致を確かめることができる．

4.2.5　擬似原像攻撃の手順

ハッシュ値 h_i が与えられたときの 54 ステップ SHA-0 の圧縮関数に対する擬似原像攻撃の手順を以下に示す．

手順 1 IS 及び PM を行え，またパディングを満たすように中間値 $\tau_j (j = 32, 33, ..., 38)$ 及びメッセージワード $m_s (s = 0, 1, ..., 15)$ の値を固定する．ただしメッセージワード m_s のうち，中立ワードとして使う部分は固定値にしない．

手順 2 8 ビットの中立ワード m_A の取りうる全ての値を用いて以下を計算する．

(a) 節 4.2.3 で示したように R_{32} から R_{38} までを計算し，p_{39} を求める．

表 4.4 : Partial Matching において計算可能なビット位置及び候補の数

j	a_j	b_j	c_j	d_j	e_j	a_j の候補数
2	all	all	all	all	all	
3	0-14,20-31	all	all	all	all	2
4	0-19,27-31	0-14,20-31	all	all	all	$2 * \frac{5}{4}$
5	0-14,20-24	0-19,27-31	0-12,18-31	all	all	$2^2 * \frac{5}{4}$
6	$9-12$	0-14,20-24	0-17,25-31	0-12,18-31	all	$2^2 * \frac{5}{4} * \frac{17}{16}$
6	$9-25$	$11-25$	?	?	?	
7	$9-25$	$9-25$	$9-23$	?	?	
8	$9-25$	$9-25$	$7-23$	$9-23$	?	
9	$4-25$	$9-25$	$7-23$	$7-23$	$9-23$	$2^2 * \frac{33}{32} * 3^3$
10	$2-25$	$4-25$	$7-23$	$7-23$	$7-23$	$2^2 * \frac{33}{32} * 3^2$
11	$2-25$	$2-25$	$2-23$	$7-23$	$7-23$	$2^2 * \frac{33}{32} * 3$
12	$2-25$	$2-25$	$0-23$	$2-23$	$7-23$	$2^2 * \frac{33}{32}$
13	all	$2-25$	$0-23$	$0-23$	$2-23$	2
14	all	all	$0-23$	$0-23$	$0-23$	1
15	all	all	all	$0-23$	$0-23$	1
16	all	all	all	all	$0-23$	1
17	all	all	all	all	all	

(b) $p_{j+1} \leftarrow R_j(p_j, w_j)(j = 39, 40, ..., 53)$,$p_0 \leftarrow h_i - p_{54}, p_{j+1} \leftarrow R_j(p_j, w_j)(j = 0, 1)$ を計算する．

(c) a_3 の 0-14,20-31 ビットを 2 通り求める．a_4 の 0-19,27-31 ビットを $2 * \frac{5}{4}$ 通り求める．a_5 の 0-14,20-24 ビットを $2^2 * \frac{5}{4}$ 通り求める．a_6 の 9-12 ビットを $2^2 * \frac{5}{4} * \frac{17}{16}$ 通り求める．

(d) テーブル $(m_A, p_2, a_3, a_4, a_5, a_6)$ を作る．1 つの値の m_A につき，このテーブルは $2^2 * \frac{5}{4} * \frac{17}{16}$ 個の要素を持つ．2^8 通りの m_A を用いて計算を行っているので，テーブルは $2^{10} * \frac{5}{4} * \frac{17}{16}$ 個の要素を持つ．

手順 3 5 ビットの中立ワード m_B の取りうる全ての値を用いて以下を計算する．

(a) 節 4.2.3 で示したように R_{38} から R_{32} までを計算し，p_{32} を求める．

(b) $p_j \leftarrow R_j^{-1}(p_{j+1}, w_j)(j = 31, 30, ..., 17)$ を計算する．

(c) e_{16}, e_{15}, e_{14} の 0-23 ビットを求める．e_{13} の 2-23 ビットを 2 通り求める．e_{12} の 7-23 ビットを $2^2 * \frac{33}{32}$ 通り求める．e_{11} の 7-23 ビットを $2^2 * 3 * \frac{33}{32}$ 通り求める．e_{10} の 7-23 ビットを $2^2 * 3^2 * \frac{33}{32}$ 通り求める．e_9 の 9-23 ビットを $2^2 * 3^3 * \frac{33}{32}$ 通り求める．

(d) CF_A,CF_B それぞれの計算結果を比較し，a_6 の 9-12 ビットと a_5 の 11-14,20-24 ビット，合計 13 ビットの値が一致するか確認する．

(e) 一致した場合，そのときの m_A,m_B のペアを用いて p_3 から p_6 までを計算する．1 ステップ計算するごとに，繰り上がりが正しいかどうかの確認を行う．また p_5 を計算すると a_5 で新たに 6 ビット分の一致を確かめられるようになる (ビット位置 15-19,25) ので確認する．また p_6 を計算すると a_6 で新たに 13 ビット分の一致を確かめられるようになる (ビット位置 13-25) ので確認する．

(f) ここまでの確認を通った場合，p_{16} から p_6 までを計算する．このとき繰り下がりが正しいかどうかを確認する．p_6 の 160 ビットの値が一致しているかを確かめる．

(g) 一致した場合，そのときの入力メッセージワードと p_0 が擬似原像である．

4.2.6 計算量評価

今回の攻撃の計算量評価を行う．なお，1 ステップの計算の計算量を $\frac{1}{54}$ とする．

手順 1 無視できる

手順 2a $2^8 * \frac{7}{54}$

手順 2b $2^8 * \frac{17}{54}$

手順 2c $\frac{632}{9}(= 2^8 * \frac{1}{54} * (2 + 2 * \frac{5}{4} + 2^2 * \frac{5}{4} + 2^2 * \frac{5}{4} * \frac{17}{16}))$

手順 3a $2^5 * \frac{7}{54}$

手順 3b $2^5 * \frac{15}{54}$

手順 3c $\frac{2720}{27}(= 2^5 * \frac{1}{54} * (1 + 1 + 1 + 2 + 2^2 * \frac{33}{32} + 2^2 * 3 * \frac{33}{32} + 2^2 * 3^2 * \frac{33}{32} + 2^2 * 3^3 * \frac{33}{32}))$

CF_A は $2^2 * \frac{5}{4} * \frac{17}{16}$ 通りの値を，CF_B は $2^2 * 3^3 * \frac{33}{32}$ 通りの値を算出する．手順 3d で 13 ビットの一致を確かめた後，残るペア数は $2^{-7} * 75735 (= 2^2 * \frac{5}{4} * \frac{17}{16} * 2^2 * 3^3 * \frac{33}{32} * 2^{-13})$ 個である．

手順 3e 計算量 $2^{-7} * 75735 * \frac{1}{54}$ で p_3 を計算できる．繰り上がりが正しいか確認すると残りペア数は $2^{-8} * 75735$ になる．計算量 $2^{-8} * 75735 * \frac{1}{54}$ で p_4 を計算できる．繰り上がりの確認により残りペア数は $2^{-8} * 75735 * \frac{4}{5}$ になる．計算量 $2^{-8} * 75735 * \frac{4}{5} * \frac{1}{54}$ で p_5 を計算できる．繰り上がりと，6 ビットの一致の確認により，残りペア数は $2^{-15} * 75735 * \frac{4}{5}$ になる．計算量 $2^{-15} * 75735 * \frac{4}{5} * \frac{1}{54}$ で p_3 を計算できる．繰り上がりと，13 ビットの一致の確認により，残りペア数は $2^{-28} * 75735 * \frac{4}{5} * \frac{16}{17}$ になる．

手順 3f ここまでの残りペア数が非常に少なく，そのため計算量も非常に小さくなるのでここの計算量は無視できる．繰り下がりの確認により，残りペア数は $2^{-19} (= 2^{-28} * 75735 * \frac{4}{5} * \frac{16}{17} * 2^{-2} * 3^{-3} * \frac{32}{33})$ となる．すでに 32(=13+6+13) ビットの一致を確かめているので，残り 128 ビットを一致を確かめる．残りペア数は $2^{-147} (= 2^{-19} * 2^{-128})$ となる．よってこれまでの計算を 2^{147} 回繰り返せば擬似原像が求まると期待できる．

これまでの計算にかかった計算量は，手順 2 から手順 4 までの計算量を全て足して，$2^8 * \frac{7}{54} + 2^8 * \frac{17}{54} + \frac{632}{9} + 2^5 * \frac{7}{54} + 2^5 * \frac{15}{54} + \frac{2720}{27} + 2^{-7} * 75735 * \frac{1}{54} + 2^{-8} * 75735 * \frac{1}{54} + 2^{-8} * 75735 * \frac{4}{5} * \frac{1}{54} + 2^{-15} * 75735 * \frac{4}{5} * \frac{1}{54} = 2^{8.31\ldots} \leq 2^{8.4}$ となる．よって擬似原像は $2^{8.4} * 2^{147} = 2^{155.4}$ 程度の計算量で求まり，原像は $2^{158.7} (= 2 * 2^{(160+155.4)/2})$ 程度の計算量で求まる．なお，メモリは主に手順 2d で使用され，メモリ量は 2^{11}words 程度である．

4.3 54 ステップ SHA-1 への攻撃

IS において，Aoki ら高々4 ステップの IS しか構築できなかった．我々は SHA-1 と同様に中間値のビットごとに同様に独立性を考えた結果，7 ステップ以上の IS が構築可能であることを発見した．中間一致攻撃を行う場合，関数をどのように分けるかが重要である．Aoki らは [5] で SHA-0 と同様に SHA-1 の関数分けの方法を述べている．Aoki らは IS と PM を行うステップ数の合計が 18 のとき 48 ステップの SHA-1 に攻撃が行え，ステップ数の合計が 22 のとき 54 ステップの SHA-1 に対して攻撃が行えることを発見した．また，Aoki らは 4 ステップの IS と 14 ステップの PM を構築し，IS と PM の合計ステップ数を 18 として 48 ステップの SHA-1 に対して攻撃を行なった．本論文では 7 ステップの IS と 15 ステップの PM を構築し，IS と PM の合計ステップ数を 22 として 54 ステップの SHA-1 に対して攻撃を行うことができることを示す．

4.3.1 関数分けおよび中立ワード

CF_B と独立な中立ワードを m_A，CF_A と独立な中立ワードを m_B とする．また，中立ワード以外のメッセージワードを m_C とする．表 4.5 で w_j が m_A と m_B の影響をどのように受けるかを示す．例えば w_4=$m_A \oplus m_A^{\gg 2} \oplus m_B \oplus m_C$ である．複数のステップで w_j がどちらの中立ワードの影響も受けるが，中立ワードとして使うビット位置が重複していないので，これらの中立ワードは独立に動かせる．

また，表 4.5 では関数の分け方も示す．

なお，m_{13} の 0 ビット目を 1，m_{15} の 0-8 ビットを 10 進数で 447 と固定値にしている．これは擬似原像攻撃を原像攻撃に変換する際にパディングを満たすためであり，満たす方法は [9] を参照．

4.3.2 Initial Structure

図 4.4 のように 7 ステップの IS を構築した．目標は p_9 の値が m_B に依存しない，かつ，p_2 の値が m_A に依存しないようにすることである．R_2 から R_8 まで計算したときの m_A による丸め差分と R_8 から R_2 まで m_B を用いて逆算したときの丸め差分のビット位置が重複しないようにできれば目標が達成できる．

なお，τ_j は 32 ビットの中間値を表している．中間値 τ_j のうち，いくつかのビットを固定値にしている．固定値にしたビットと値を表 4.6 に示す．また，節 4.2.3 と同様，加算を分けており，表 4.7 に加算をどのように分けたかを示す．

まず m_A による丸め差分パスを考える．

ステップ 2　$a_3 \leftarrow \tau_2 + w_2^1$ を計算する．このとき τ_2 の値によっては繰り上がりによって a_3 の 31 ビット目に丸め差分が入る場合があるが，τ_2 の 21-31 ビットを **0** にしているので繰り上がりは発生せず，a_3 の 23-30 ビットに丸め差分が入る．

j	m_A の影響	m_B の影響	
0	0	0	CF_B
1	0	0	↑
2	$m_A^{\gg 1}$	m_B	
3	0	0	
4	$m_A \oplus m_A^{\gg 2}$	m_B	
5	m_A	0	IS
6	$m_A \oplus m_A^{\gg 2}$	m_B	
7	0	0	
8	m_A	m_B	
9	0	0	↓
10	$m_A \oplus m_A^{\gg 1}$	0	
11	0	0	
12	m_A	0	
13	m_A	0	
14	$m_A^{\gg 1}$	0	
15	0	0	CF_A
16	m_A	0	
17	m_A	0	
18	$m_A^{\gg 1}$	0	
19	0	0	
20	0	0	
21	m_A	0	
22	$m_A \oplus m_A^{\gg 1}$	0	
23	0	0	

j	m_A の影響	m_B の影響	
24	m_A	$m_B^{\ll 1}$	↓
25	m_A	0	
26	0	0	
27	0	$m_B^{\ll 2}$	
28	m_A	0	
29	0	0	
30	0	$m_B^{\ll 3}$	
31	0	0	PM
32	m_A	$m_B^{\ll 2}$	
33	0	$m_B^{\ll 4}$	
34	m_A	0	
35	0	0	
36	m_A	$m_B^{\ll 5}$	
37	0	0	
38	m_A	$m_B^{\ll 2} \oplus m_B^{\ll 4}$	↑
39	0	$m_B^{\ll 6}$	
40	0	$m_B^{\ll 2} \oplus m_B^{\ll 3}$	
41	0	0	
42	0	$m_B^{\ll 7}$	
43	0	$m_B^{\ll 4}$	
44	0	$m_B^{\ll 4} \oplus m_B^{\ll 6}$	
45	0	$m_B^{\ll 8}$	CF_B
46	0	$m_B^{\ll 4}$	
47	0	0	
48	0	$m_B^{\ll 4} \oplus m_B^{\ll 9}$	
49	0	0	
50	0	$m_B^{\ll 6} \oplus m_B^{\ll 8}$	
51	0	$m_B^{\ll 10}$	
52	0	$m_B^{\ll 3} \oplus m_B^{\ll 6} \oplus m_B^{\ll 7}$	
53	0	0	↑

表 4.5：54 ステップ SHA-1 の関分け

ステップ 3　$a_4 \leftarrow \tau_3 + a_3^{1\lll 5}$ を計算する．このとき τ_3 の値によっては繰り上がりによって a_4 の 4 ビット目に丸め差分が入る場合があるが，τ_3 の 0-15 ビットを **0** にしているので繰り上がりは発生せず，a_4 の 0-3,28-31 ビットに丸め差分が入る．

ステップ 4　$a_5 \leftarrow \tau_4 + w_4^1 + F_1^1(b_4, c_4, d_4) + a_4^{1\lll 5}$ を計算する．このとき $\tau_4 + F_1^1(b_4, c_4, d_4) = \mathbf{0}$(0-21 ビット) としているので，$a_5$ の 9 ビット目には繰り上がりは発生せず，a_5 の 1-8,22-31 ビットに丸め差分が入る．

ステップ 5　$a_6 \leftarrow \tau_5 + w_5 + K_1 + F_1^1(b_5, c_5, d_5) + a_5^{\lll 5}$ を計算する．$w_5 = m_{\_}K_1$(0-23 ビット) とすれば，$\tau_5 + w_5 + K_1 = \mathbf{0}$(0-13 ビット) となり，$a_6$ の 14 ビット目には繰り上がりは発生せず，a_5 の 0-13,21-31 ビットに丸め差分が入る．

ステップ 6　$a_7 \leftarrow \tau_6 + w_6^1 + K_1 + F_1^1(b_6, c_6, d_6) + a_6^{\lll 5}$ を計算する．a_7 にどのような丸め差分が出ても IS は構築可能であるので，丸め差分を考慮しない．

ステップ 7　$a_8 \leftarrow e_7 + \tau_7 + w_7 + F_1^1(b_7, c_7, d_7) + K_1 + a_7^{\lll 5}$ を計算する．a_8 に入る丸め差分は考慮しない．

ステップ 8　$a_9 \leftarrow e_8 + \tau_8 + w_8^1 + F_1(b_8, c_8, d_8) + K_1 + a_8^{\lll 5}$ を計算する．a_9 に入る丸め差分は考慮しない．

次に m_B による丸め差分パスを考える．

ステップ 8　$e_8 \leftarrow \tau_8 - w_8^2$ を計算する．このとき τ_8 の値によっては繰り下がりによって e_8 の 19 ビット目に丸め差分パスが入る場合があるが，τ_{38} の 14-18 ビットを **1** にしているので繰り下がりは発生せず，e_8 の 14-18 ビットに丸め差分が入る．

ステップ 7　$e_7 \leftarrow \tau_7 - F_1^2(b_7, c_7, d_7)$ を計算する．τ_7 の 14-18 ビットを **1** にしているので e_7 の 19 ビット目に繰り下がりは発生せず，e_7 の 14-18 ビットに丸め差分が入る．

ステップ 6　$e_6 \leftarrow \tau_6 - F_1^2(b_6, c_6, d_6) - w_6^2$ を計算する．τ_6 の値によらず，e_6 の 19 ビット目には繰り下がりが発生し，丸め差分が入る．これ以上の丸め差分の拡大を抑えるため e_6 の 20 ビット目に繰り下がりが発生しないようにする．τ_6 の 14-19 ビットを **1** にしているので e_6 の 20 ビット目に繰り下がりは発生せず，e_6 の 14-19 ビットに丸め差分が入る．

ステップ 5　$e_5 \leftarrow \tau_5 - F_1^2(b_5, c_5, d_5)$ を計算する．τ_5 の 14-20 ビットを **1** にしているので e_5 の 21 ビット目に繰り下がりは発生せず，e_5 の 14-20 ビットに丸め差分が入る．

ステップ 4　$e_4 \leftarrow \tau_4 - K_1 - F_1^2(b_4, c_4, d_4) - w_4^2 + a_4^{2\lll 5}$ を計算する．e_4 に入る丸め差分は考慮しない．

表 4.6： τ_j の固定値にするビット位置及び値

ステップ j	τ_j
2	0-15,21-31=**0**
3	0-15,21-31=**0**
4	0-31=**0**
5	0-13,21-31=**0** 14-20=**1**
6	0-13,20-31=**0** 14-19=**1**
7	14-18=**1**
8	14-18=**1**

ステップ 3　$e_3 \leftarrow a_4 - \tau_3 - K_1 - F_1(b_3, c_3, d_3) - w_3 + a_3^{2\lll 5}$ を計算する．e_3 に入る丸め差分は考慮しない．

ステップ 2　$e_2 \leftarrow a_3 - \tau_2 - K_1 - F_1(b_2, c_2, d_2) - w_2^2 + a_2^{\lll 5}$ を計算する．e_2 に入る丸め差分は考慮しない．

a_3 や a_4 には m_A,m_B による丸め差分がどちらも入っているが，丸め差分のビット位置は重複していない．よって m_A と m_B の丸め差分のビット位置はひとつも重複しない．これで目標は達成でき，IS が構築できた．

4.3.3　Partial Matching

15 ステップを部分的に独立に計算し，13 ビットだけ中間値の一致を確かめられるような PM を適用した．CF_A の計算で p_{24} を求め，CF_B の計算で p_{39} を求めており，これらの中間値から p_{28} を部分的に求めて一致を確かめる．表 4.8 に今回の Partial Matching で，どの部分を計算したかを示す．

CF_A 側から p_{28} を求めていく．

a_{25} の計算　$a_{25} \leftarrow e_{24} + K_2 + F_2(b_{24}, c_{24}, d_{24}) + a_{24}^{\lll 5} + w_{24}$ を計算したいが，w_{24} の 15-19 ビットは本来 m_B の影響を受けているため，値がわからない．そのため a_{25} は 0-14,20-31 ビットのみが部分的に計算できる．このとき 20-31 ビットが 19 ビット目からの繰り上がりの影響を受けるか受けないかわからないので，繰り上がる場合と繰り上がらない場合のどちらの a_{25} の値も候補に入れて考える．このステップで候補は 2 倍になる．

a_{26} の計算　a_{26} を計算するとき，$a_{25}^{\lll 5}$ の加算によって 20-24 ビットに未知の値が入る，a_{26} は 0-19,25-31 ビットのみが部分的に計算できる．このうち 0-19,27-31 ビットの値のみを用いて PM を行っていきたい．すると 27-31 ビットへの繰り上がりを次のように考えることができる：繰り上がりを考えずに 25,26 ビットを求める．求めた値が”11”でない場合は 27-31 ビットに繰り上がりは入らない．”11”のときだけ繰り上がる場合と繰り上がらない場合のどちらの

表 4.7 : 加算の分け方

	w_j		F		a_j	
ステップ j	w_j^1	w_j^2	F_2^1	F_2^2	a_j^1	a_j^2
2	23-31	0-22	–		–	
3	–		–		0-15,21-31	16-20
4	22-31	0-21	23-31	0-22	0-15,21-31	16-20
5	–		0-3,21-31	4-20	–	
6	0-13,19-31	14-18	0-13,19-31	14-18	–	
7	–		0-13,19-31	14-18	–	
8	0-13,19-31	14-18	–		–	

値も候補に入れて考えるとすると，このステップでは $\frac{1}{4}$ の確率で候補が 2 倍になる．平均するとこのステップでは候補が $\frac{5}{4}$ 倍になる．

a_{27} の計算　a_{27} を計算するとき，w_{26} によって 25-31 ビットに未知の値が入り，論理関数 F_2 によって 15-19 ビットに未知の値が入る．a_{27} は 0-14,20-24 ビットのみが部分的に計算できる．20-24 ビットへの繰り上がりを考えると，このステップでは候補が 2 倍になる．

a_{28} の計算　a_{28} を計算するとき，論理関数 F_2 によって 13-24 ビットに未知の値が入り，$a_{27}^{\lll 5}$ の加算によって 0-4,20-24,30-31 ビットに未知の値が入る．a_6 は 5-12,25-29 ビットのみが部分的に計算可能だが，このうち 9-12 ビットの値のみを用いて PM を行いたい．9-12 ビットへの繰り上がりをアイデア 1 を用いて考えると，このステップでは候補が平均して $\frac{17}{16}$ 倍になる．

CF_B 側から p_{28} を求めていく．

e_{38}, e_{37}, e_{36} の計算　e_{38}, e_{37}, e_{36} は全て 0-23 ビットのみが部分的に計算できる．繰り下がりが発生することはないので候補は増えない．

e_{35} の計算　e_{35} を計算するとき，論理関数 F_2 によって 0-1,24-31 ビットに未知の値が入る．よって 2-23 ビットのみが部分的に計算でき，繰り下がりにより候補は 2 倍になる．

e_{34} の計算　まず $\tau_{34} \leftarrow a_{35} - K_2 - w_{34} - F_2(b_{34}, c_{34}, d_{34})$ を計算する．このとき論理関数 F_2 によって 0-1,24-31 ビットに未知の値が入り，τ_{34} は 2-23 ビットが計算可能である．このうち 7-23 ビットのみを以降の計算で使うとすると，アイデア 1 より候補は $\frac{33}{32}$ 倍になる．また，$e_{34} \leftarrow \tau_{34} - a_{34}^{\lll 5}$ を計算すると，e_{34} の 7-23 ビットを計算できる．このとき繰り下がりを考えると候補は 2 倍になる．

e_{33}, e_{32}, e_{31} の計算　まず $\tau_{33} \leftarrow a_{34} - K_2 - w_{33}$ を計算する．a_{34} は 2-25 ビットの値しかわからないため，普通に計算した場合 2 回繰り下がりを考えなければならない．そこでメッセージワード $w_{33} = m_{33} = -K_2$ となるように 0-1 ビットを固定値にしておけば，$-K_2 - w_{33} =$ "00"(0-1 ビット) となり，繰り下がりを考えなくて済む．また，$e_{33} \leftarrow \tau_{33} - F_2(b_{33}, c_{33}, d_{33}) - a_{33}^{\lll 5}$ を

表 4.8 : Partial Matching において計算可能なビット位置及び候補の数

j	a_j	b_j	c_j	d_j	e_j	a_j の候補数
24	all	all	all	all	all	
25	0-14,20-31	all	all	all	all	2
26	0-19,27-31	0-14,20-31	all	all	all	$2 * \frac{5}{4}$
27	0-14,20-24	0-19,27-31	0-12,18-31	all	all	$2^2 * \frac{5}{4}$
28	$9-12$	0-14,20-24	0-17,25-31	0-12,18-31	all	$2^2 * \frac{5}{4} * \frac{17}{16}$
28	$9-25$	$11-25$	?	?	?	
29	$9-25$	$9-25$	$9-23$	?	?	
30	$9-25$	$9-25$	$7-23$	$9-23$	?	
31	$4-25$	$9-25$	$7-23$	$7-23$	$9-23$	$2^2 * \frac{33}{32} * 3^3$
32	$2-25$	$4-25$	$7-23$	$7-23$	$7-23$	$2^2 * \frac{33}{32} * 3^2$
33	$2-25$	$2-25$	$2-23$	$7-23$	$7-23$	$2^2 * \frac{33}{32} * 3$
34	$2-25$	$2-25$	$0-23$	$2-23$	$7-23$	$2^2 * \frac{33}{32}$
35	all	$2-25$	$0-23$	$0-23$	$2-23$	2
36	all	all	$0-23$	$0-23$	$0-23$	1
37	all	all	all	$0-23$	$0-23$	1
38	all	all	all	all	$0-23$	1
39	all	all	all	all	all	

計算すると，e_{33} は 7-23 ビットを計算できる．ただし論理関数 F_2 と $a_{33}^{\lll 5}$ によって e_{33} の 7-23 ビットは 2 回繰り下がりを考えなければならない．このとき，1 回も繰り下がりが入らない場合，1 回だけ繰り下がりが入る場合，2 回とも繰り下がりが入る場合の 3 つの候補を考える．すると候補は 3 倍になる．e_{32} の計算も e_{33} の計算と似ており，$w_{32} = m_{32} = -K_2$(0-1 ビット) とすれば 2 回の繰り下がりを考えて，e_{32} の 7-23 ビットを計算できる．アイデア 2 を用いて考えれば，候補は 3 倍になる．e_{31} も同様に，$w_{31} = m_{31} = -K_2$(0-1 ビット) とすれば 2 回の繰り下がりを考えて，e_{31} の 9-23 ビットを計算でき，候補は 3 倍に増える．

ここまで計算を行えば，a_{27}=$e_{32}^{\lll 30}$，a_{28}=$e_{31}^{\lll 30}$ は簡単に計算でき，a_{28} の 9-12 ビットと a_{27} の 11-14,20-24 ビットで一致を確かめることができる．

4.3.4 擬似原像攻撃の計算量評価

54 ステップ SHA-0 の攻撃と同様，7 ステップの確率 1 の IS を用いており，また，PM については表 4.4 及び表 4.8 を見ると，54 ステップ SHA-0 への攻撃と全く同じである．また，中立ワード m_A,m_B の取りうるビット数も同じである．よって関数の分け方による違いを除けば，この攻撃は 54 ステップ SHA-0 と同様の手順で攻撃でき，また，計算量もほぼ同じになる．

よって 54 ステップ SHA-0 への攻撃と同様，擬似原像は $2^{155.4}$ 程度の計算量で求まり，原像は $2^{158.7}$ 程度の計算量で求まる．なお，メモリ量は 2^{11}words 程度である．

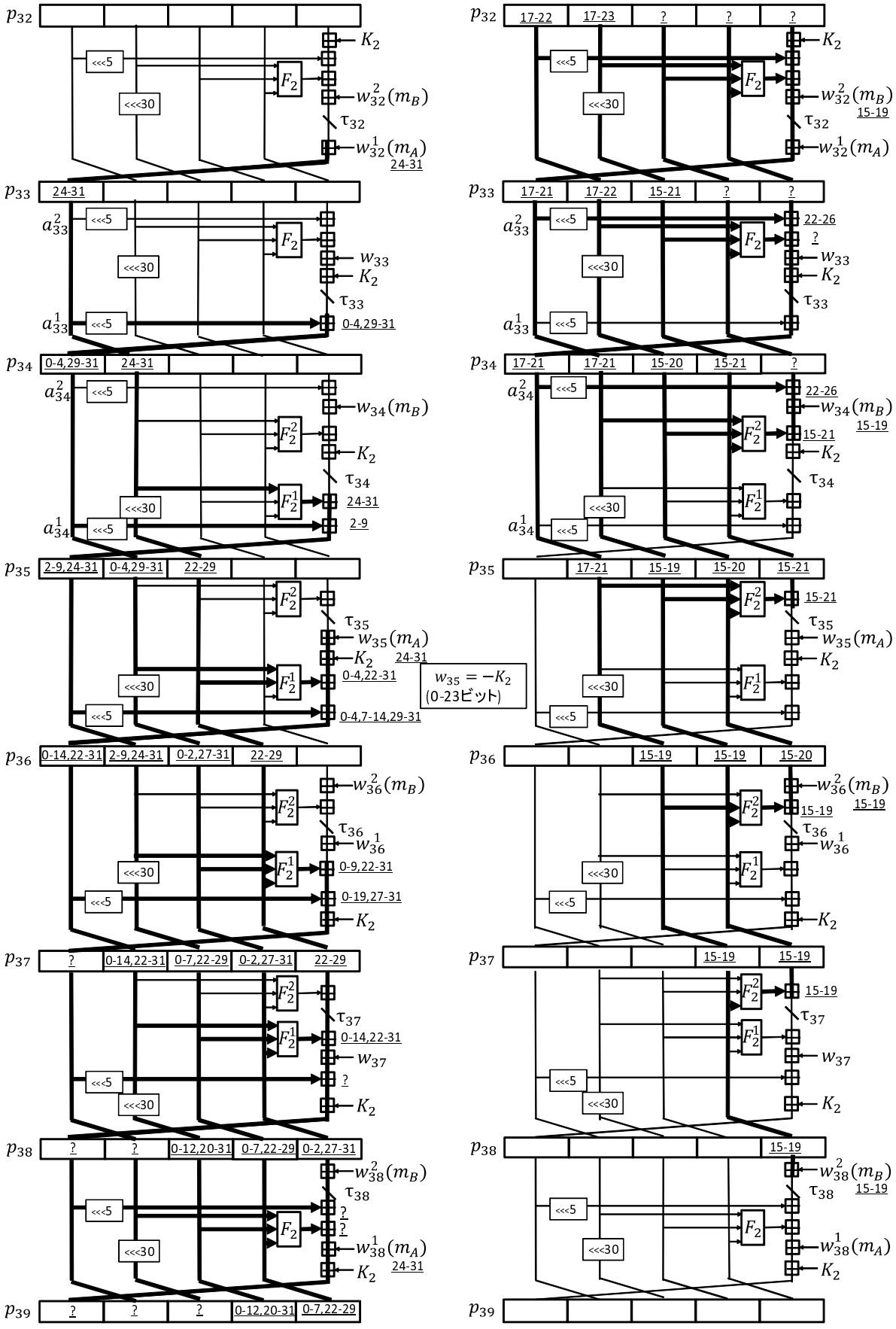

図 4.3：54 ステップ SHA-0 への攻撃で構築した 7 ステップ Initial Structure

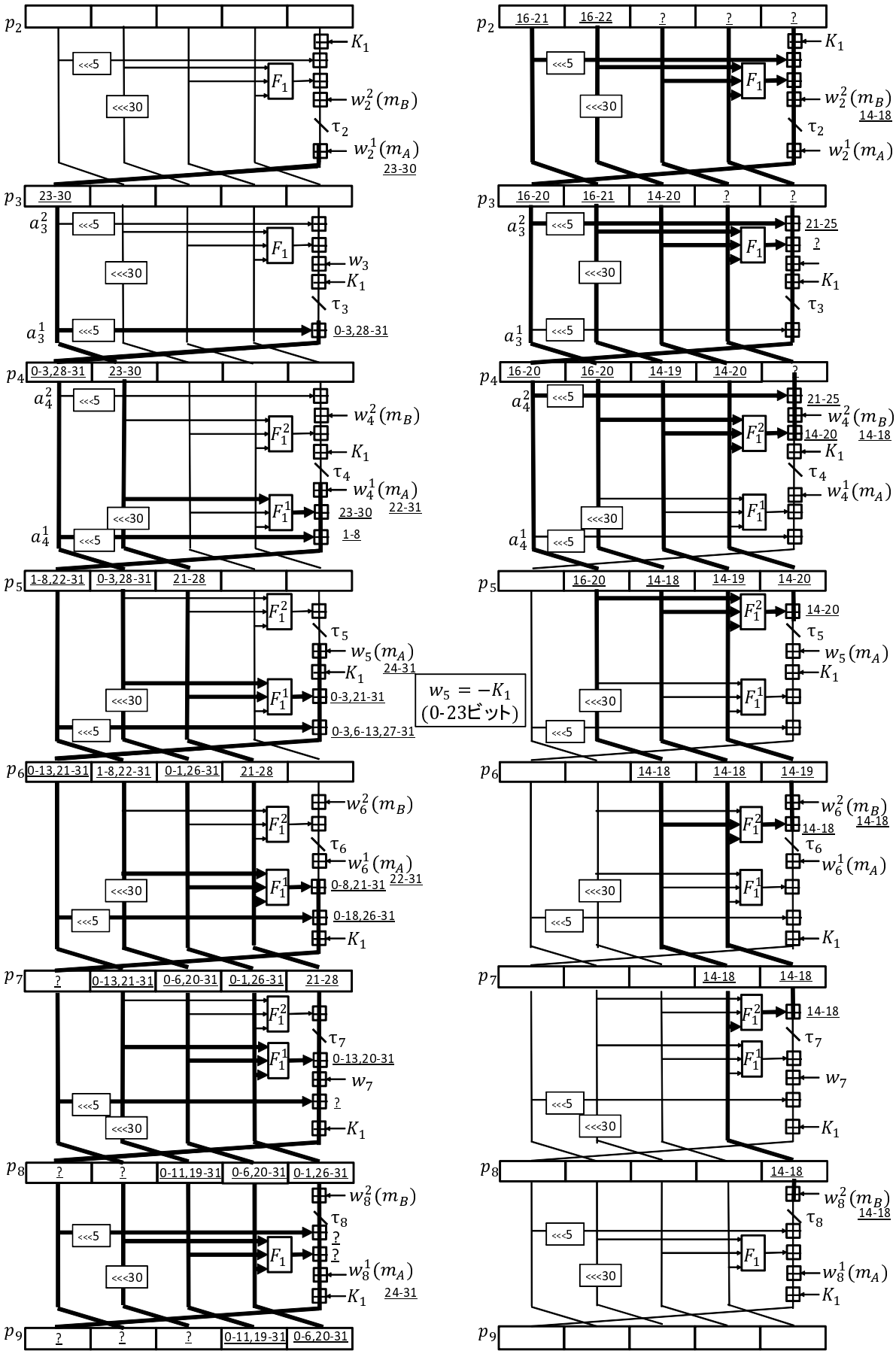

図 4.4：54 ステップ SHA-1 への攻撃で構築した 7 ステップ Initial Structure

第5章

成果2：メモリ量の劇的な削減

図 5.1 において m_A から CF_A を，m_B から CF_B を独立に計算したいが，この状態で計算を行おうとすると問題箇所にてそれぞれの計算の影響が重複するため攻撃が難しい．なお，問題箇所は r ビットの計算を行うとするこの問題に対し,Sasaki らと本研究はそれぞれ違う IS を構築して解決した．また，それぞれ構築した IS を用いて全ステップの MD5 及び全ステップの 4-passHAVAL に対して攻撃を試みた結果，Sasaki らは MD5 への攻撃は 2^{45}words，4-passHAVAL への攻撃は 2^{64}words のメモリが必要であるとしたが，本研究では MD5 への攻撃は 2^{13}words，4-passHAVAL への攻撃は 2^{30}words と，佐々木らの攻撃よりも非常に少ないメモリ量での攻撃に成功した．

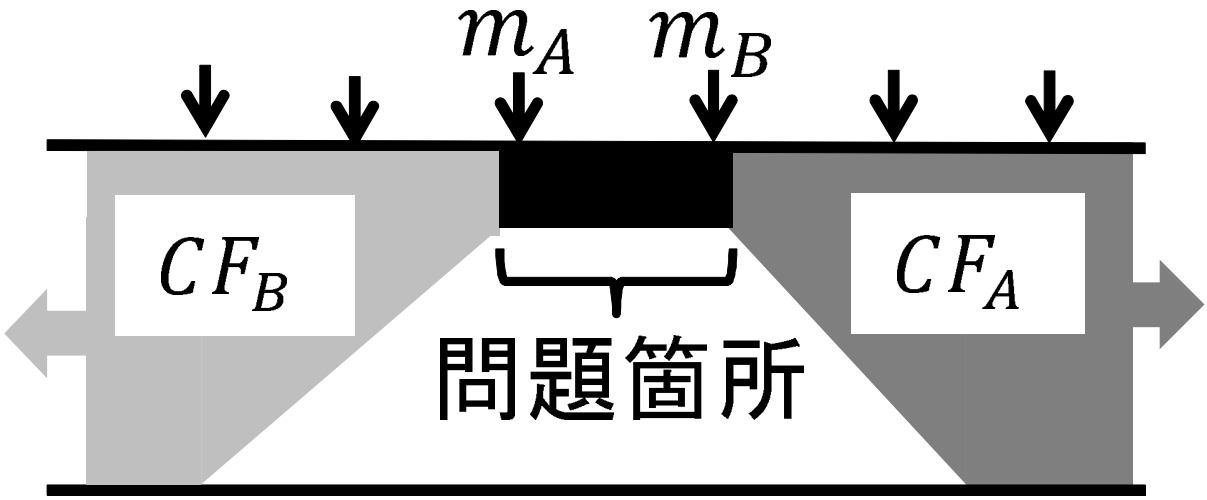

図 5.1：問題箇所

5.1 既存研究の IS:独立計算できない IS

Sasaki らは図 5.1 の問題箇所を独立に計算できるようにはしなかった．その代わりにメモリを非常に多く使用することで，図 5.1 の問題箇所を独立に計算できるようにせずとも攻撃が可能となる，Local Collision と呼ばれる手法を用いて攻撃を考えた．

5.1.1 Local Collision

ここではLocal Collisionと呼ばれる手法を紹介する．図5.1のように部分的に独立計算ができない箇所があっても，メモリを多く使用することで中間一致攻撃を成立させる手法である．

簡単のため問題箇所の計算は，m_Aとm_Bが加算されるだけの計算であると仮定する．つまり，計算結果を図5.2のようにX, Yと仮定しすると，問題箇所の計算は$Y + m_A + m_B = X$である．中間一致攻撃を行う．攻撃に必要な計算量は節3.2と変わらないが，攻撃に必要なメモリ量が節3.2の2^rから2^{2r}と，2^r倍になっている．

以下，Local Collisionの行い方を述べる．

手順1　m_Cをランダムな値に固定する．

手順2　2^{2r}通りの(X, m_A)を用いて中間値p_t=$CF_A(X, m_A)$を2^{2r}回計算し，2^{2r}個の(p_t, X, m_A)の値をテーブルTに保存する．

手順3　2^{2r}通りの(Y, m_B)を用いて中間値p_t=$CF_A(Y, m_B)$を2^{2r}回計算し，計算した中間値p_tがテーブルTに含まれているp_tと一致するかどうかチェックする，含まれていれば手順4へ．

手順4　計算結果X, Yの仮定が正しいかチェックする．つまり$Y + m_A + m_B = X$が成り立つかどうかチェックする．成り立てばそのときのM_{i-1}=$m_A \| m_B \| m_C$が圧縮関数の原像となる．rビットの一致を確かめることになるので，成り立つ確率は2^{-r}である．

手順5　手順3で一致しなかった場合，手順1に戻る．

CF_AもしくはCF_Bの計算を計算量$\frac{1}{2}$と考える．手順4までを行うと，2^{2r}の計算量で，2^{4r}回の一致を確かめられる．また，手順3及び手順4の両方で一致が成功する確率は$2^{-(n+r)}$であり，手順4までの計算を2^{n-3r}回繰り返すと，2^{n+r}回の一致が確かめられるので，圧縮関数の原像が求まる．結局$2^{n-r}(= (2^{2r}) * (2^{n-3r}))$回の計算で圧縮関数の原像が求まる．また，メモリは手順2で主に使われ，2^{2r}wordsのメモリ量が必要となる．

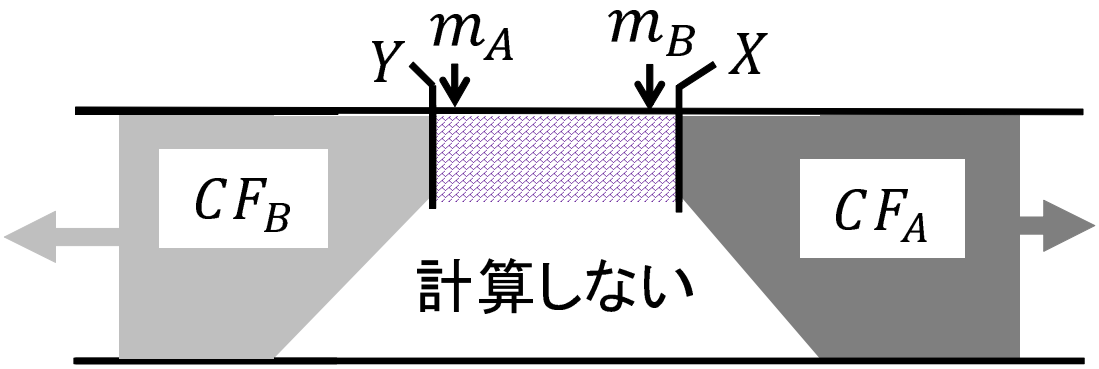

図 5.2：既存研究のIS

5.2 本研究のIS:独立計算可能な計算へ変換したIS

本研究では問題箇所を独立計算できるような等価な計算に変換することで問題を解決した．例えば図5.3のようにm_Aとm_Bの入力箇所を入れ替えることで，m_Aからの計算とm_Bからの計算の影響が重複せず，問題なく独立計算できる．この技術を使用するための条件は，問題箇所の計算を独立計算可能なものに変換しても，変換する前の計算と計算結果のつじつまが合うことである．

等価なアルゴリズムに変換した後は節3.2の攻撃手順と同じ手順で攻撃でき，攻撃に必要なメモリ量は2^rであるので，既存研究の2^{-r}のメモリ量で攻撃可能である．

等価変換の行い方であるが，例えば問題箇所の計算は$X+m_B+m_A=Y$と，単純な加算によって成り立つものだとすると，計算順序を入れ替えても問題がないという交換法則を用いることができるので，問題箇所の計算は$X+m_A=Y-m_B$と変換することができる．ここで$X+m_A=t$,$Y-m_B=t$としてtを固定すると，図5.3のように，$X=t-m_A$,$Y=t+m_B$と，Xはm_Aによって一意に決まり，Yはm_Bによって一意に決まる．また，m_Aとm_Bの計算は独立になる．

このように独立計算可能となる等価な計算に変換すれば，Local Collisionを用いることなく中間一致攻撃が行え，節3.2と同様の手順で攻撃できるためメモリ量は既存研究と比較して激減する．このように独立計算可能な等価な計算への変換が可能なパスを発見したことで攻撃に必要なメモリ量を劇的に削減した．

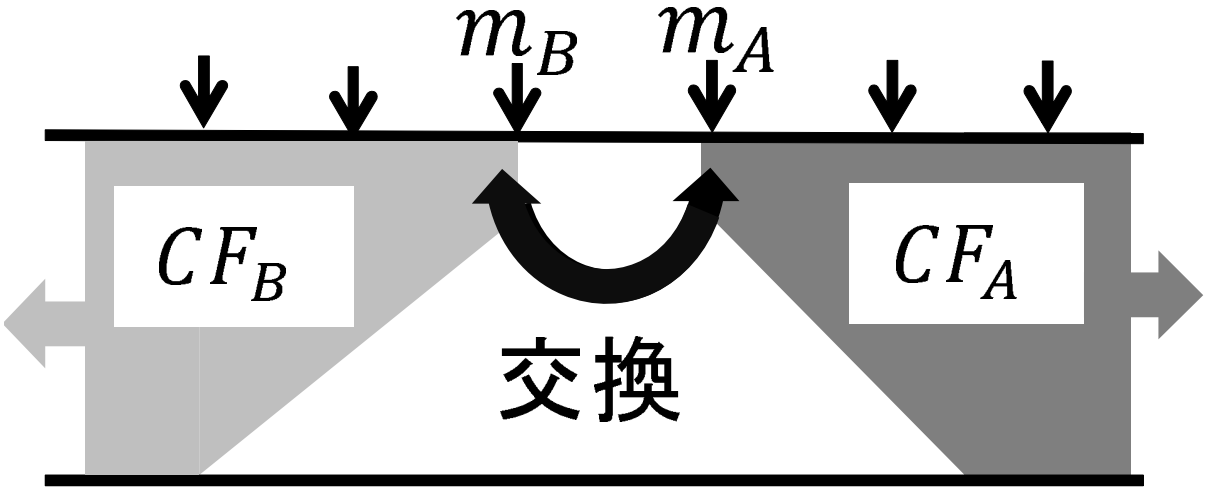

図 5.3：成果2で用いたIS

5.3 MD5への攻撃

既存研究のISの部分を改良し，[1]では2^{45}Wordsのメモリ量を使用していたのに対し，本研究では2^{13}Wordsで攻撃可能であることがわかった．以下，既存研究の攻撃と本研究の攻撃の比較を行う．

5.3.1 既存攻撃 [1]

この節では既存研究の概要について述べる．攻撃の詳細は [1] を参照されたい．既存研究では Splice-and-Cut,Initial Structure,Partial Matching を用いた中間一致攻撃を行っている．既存攻撃では表 5.1 のように圧縮関数を分けている．m_6 と m_{14} を中立ワードとし，CF_B を m_{14} と独立に，CF_A を m_6 と独立に計算している．なお，skip は PM を行うための部分計算を行う箇所である．

5.3.2 既存攻撃の IS

既存攻撃ではステップ 14 からステップ 17 まで Initial Structure を構築している．ステップ 14 で入力される m_{14} を用いて p_{18} までを m_6 と独立に計算し，ステップ 17 で入力される m_6 を用いて p_{14} を m_{14} と独立に計算したい．既存攻撃ではステップ 14 からステップ 16 までは m_{14} と m_6 の影響が重複しないようにできたが，ステップ 17 では互いの影響が重複している．ステップ 17 の計算を図 5.4 に示す．なお，$Q'_{18} = (Q_{18} - Q_{17}^{m_{14}})^{\ggg s17} - K_2$ である．また，Q_{17}, Q_{16}, Q_{15} をそれぞれ $Q_{17}^{m_{14}}, Q_{16}^{m_{14}}, Q_{15}^{m_{14}}$ と示しているが，これは Q_{17}, Q_{16}, Q_{15} の値が m_{14} の値によって決まることを示す．

Q_{14} から Q'_{18} までの計算は

$$Q_{14} + F_2(Q_{17}^{m_{14}}, Q_{16}^{m_{14}}, Q_{15}^{m_{14}}) + m_6 = Q'_{18} \tag{5.1}$$

と行われる．既存研究では，この計算を独立計算せず，節 5.1.1 のように攻撃した．つまり，(Q'_{18}, m_{14}) を用いて CF_A を計算し，(Q_{14}, m_6) を用いて CF_B を計算し，後に式の一致を確かめた．

5.3.3 既存攻撃の手順及び計算量・メモリ量

既存研究では IS と PM を行うために，Q_{14} の 5 から 16 ビット (12 ビット分)，m_6 の 21 から 28 ビット (8 ビット分)，及び m_{14} の 0 から 19 ビット (20 ビット分) を固定値にしている．よって (Q'_{18}, m_{14}),(Q_{14}, m_6) はそれぞれ 44 ビット分の値を変数として動かすことができる．

以下，節 5.1.1 と同様に簡単に手順を示し，計算量・メモリ量を概算する．なお，PM の手順及び計算量は考えない．

手順 1　m_{14}, m_6 以外のメッセージワードをランダムな値に固定する．

手順 2　2^{44} 通りの (Q'_{18}, m_{14}) を用いて中間値 p_t=$CF_A(Q'_{18}, m_{14})$ を 2^{44} 回計算し，2^{44} 個の (p_t, Q'_{18}, m_{14}) の値をテーブル T に保存する．

手順 3　2^{44} 通りの (Q_{14}, m_6) を用いて中間値 p_t=$CF_A(Q_{14}, m_6)$ を 2^{44} 回計算し，計算した中間値 p_t がテーブル T に含まれている p_t と一致するかどうかチェックする，含まれていれば手順 4 へ．

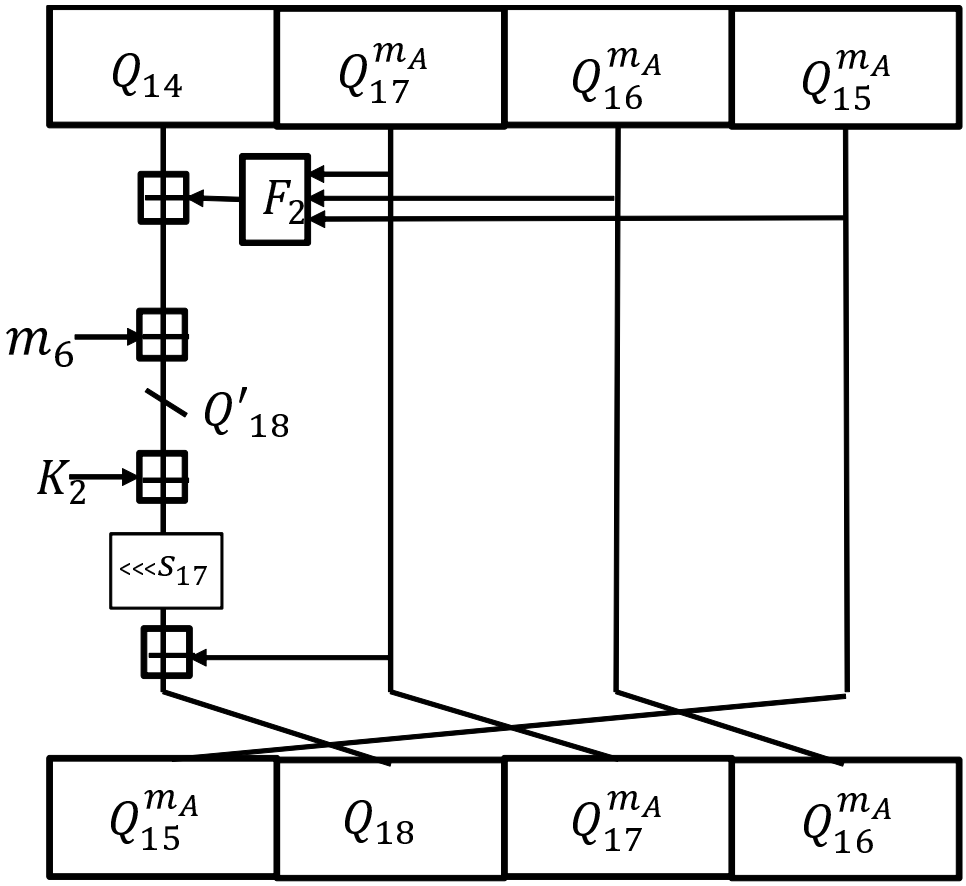

図 5.4 : 既存研究の IS の一部 (MD5 への擬似原像攻撃)

手順 4　式 $Q_{14} + F_2(Q_{17}^{m_{14}}, Q_{16}^{m_{14}}, Q_{15}^{m_{14}}) + m_6 = Q'_{18}$ が成り立つかチェックする．成り立てばそのときのメッセージワードで構築したメッセージブロック M_{i-1} が圧縮関数の原像となる．32 ビットの一致を確かめることになるので，成り立つ確率は 2^{-32} である．

手順 5　手順 3 で一致しなかった場合，手順 1 に戻る．

CF_A もしくは CF_B の計算を計算量 $\frac{1}{2}$ と考える．手順 4 までを行うと，2^{44} の計算量で，2^{88} 回の一致を確かめられる．また，手順 3 及び手順 4 の両方で一致が成功する確率は $2^{-160}(=2^{-(128+32)})$ であり，手順 4 までの計算を 2^{72} 回繰り返すと，2^{160} 回の一致が確かめられるので，圧縮関数の原像が求まる．結局 $2^{116}(=2^{44} * 2^{72})$ の計算量で圧縮関数の原像が求まる．また，メモリは手順 2 で主に使われ，2^{44}words のメモリ量が必要となる．

[1] では PM による計算量・メモリ量の増加や，計算量 1 の定義などを考慮し，擬似原像攻撃は $2^{116.9}$ 程度で求まり, メモリ量は 2^{45}words が必要であるとしている．

5.3.4　提案攻撃

提案攻撃では，図 5.5 のように既存研究では独立計算できなかったステップ 17 を独立計算可能な等価な計算に置き換えることで，メモリ量を減らした．この変更以外 (関数分け，PM,IS のステップ 17 以外の部分など) は既存研究を踏襲している．以下，提案攻撃の概要について

述べる.

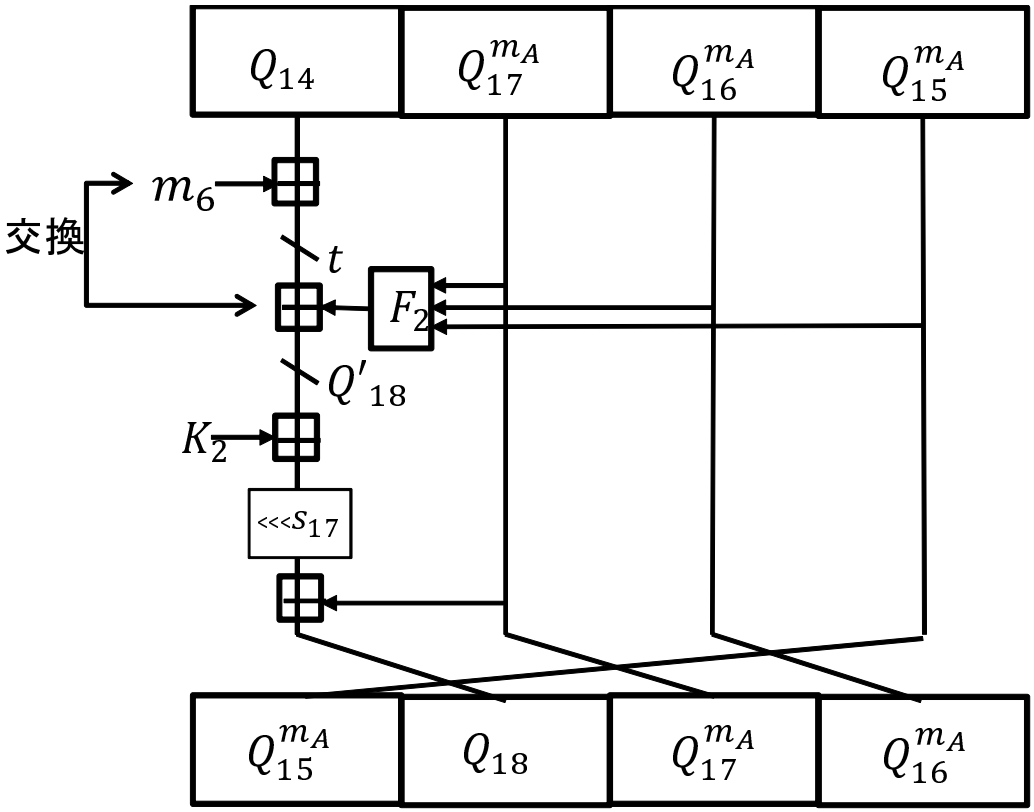

図 5.5 : 提案攻撃の IS の一部 (MD5 への擬似原像攻撃)

5.3.5 提案攻撃の IS

図 5.5 のように IS のステップ 17 を等価な計算に変換した.

つまり,

$$Q_{14} + F_2(Q_{17}^{m_{14}}, Q_{16}^{m_{14}}, Q_{15}^{m_{14}}) + m_6 = Q'_{18} \tag{5.2}$$

という計算を

$$Q_{14} + m_6 = Q'_{18} - F_2(Q_{17}^{m_{14}}, Q_{16}^{m_{14}}, Q_{15}^{m_{14}}) = t \tag{5.3}$$

と変換した．t を固定することで $Q_{14} = t - m_6$ と Q_{14} は m_6 の値によって一意に決まるようになり，また，$Q'_{18} = t + F_2(Q_{17}^{m_{14}}, Q_{16}^{m_{14}}, Q_{15}^{m_{14}})$ と，Q'_{18} も m_{14} の値によって一意に決まるようになった.

5.3.6 提案攻撃の計算量・メモリ量

既存攻撃と同様の IS と PM を行うために，m_6 の 21 から 28 ビット (8 ビット分)，及び m_{14} の 0 から 19 ビット (20 ビット分) を固定値にしている．これだけでは十分でなく，Q_{14} の 5 から 16 ビットを固定値にしなければならないが，m_6 の 5 から 16 ビット (12 ビット分) を更に固定値にすることで，$Q_{14} = t - m_6$ より，Q_{14} の 5 から 16 ビットを固定値にすることができ

る．結局 m_6 の 5 から 16 ビット，21 から 28 ビット (20 ビット分)，及び m_{14} の 0 から 19 ビット (20 ビット分) を固定値にする．よって m_{14},m_6 はそれぞれ 12 ビット分の値を変数として動かすことが出来る．

計算量及びメモリ量は，節 3.2 において $r = 12$ とすることで概算することができ，結局 2^{116} の計算量で圧縮関数の原像が求まる．また，2^{44}words のメモリ量が必要となる．既存研究の計算量・メモリ量の概算と比較すると，計算量は変わらず，メモリ量は 2^{-32} となっている．

結局，詳細な擬似原像の計算量は既存研究 [1] の $2^{116.9}$ と同じであり，必要なメモリ量は $2^1 3(=2^{45} * 2^{-32})$ である．

また，原像は $^{123.4}=2 * 2^{(128+116.9)/2}$ 程度の計算量で求まる．

5.4 4-passHAVAL への攻撃

既存研究の IS の部分を改良し，[2] では 2^{64}Words のメモリ量を使用していたのに対し，本研究では 2^{30}Words で攻撃可能であることがわかった．以下，既存研究の攻撃と本研究の攻撃の比較を行う．

5.4.1 既存攻撃 [2]

この節では既存研究の概要について述べる．攻撃の詳細は [2] を参照されたい．既存研究では Splice-and-Cut,Initial Structure,Partial Matching を用いた中間一致攻撃を行っている．既存攻撃では表 5.2 のように圧縮関数を分けている．m_5 と m_{24} を中立ワードとし，CF_B を m_{24} と独立に，CF_A を m_5 と独立に計算している．

5.4.2 既存攻撃の IS

既存攻撃ではステップ 24 からステップ 32 まで Initial Structure を構築している．この Initial Structure を図 5.6 に示す．なお，ステップ 17 の K_1 や $f_1 \circ \phi_1$ の出力を加算するときの加算の値を c_1 とすると，$Q'_{17} = Q_{17}^{\ggg 11} + c_1$ であり，また，ステップ 33 の K_2 や $f_2 \circ \phi_2$ の出力を加算するときの加算の値を c_2 とすると，$Q'_{33} = Q_{33} - c_2$ である．図 5.6 のように m_5 及び m_{24} の影響が Q'_{17} から Q'_{33} までの 1 ワード分だけの計算にしか影響しないよう，途中のステップで吸収特性 (後述) を用いて m_5 及び m_{24} の影響が他の箇所に影響するのを抑えている．

ステップ 24 で入力される m_{24} を用いて p_{33} までを m_5 と独立に計算し，ステップ 32 で入力される m_5 を用いて p_{24} を m_{24} と独立に計算したい．しかし，既存研究では Q'_{17} から Q'_{33} までの計算で図 5.6 のようにそれぞれの中立ワードの影響が重複している．

Q'_{17} から Q'_{33} までの計算は

$$(Q'_{17} + m_{24})^{\ggg 11} + m_5 = Q'_{33} \tag{5.4}$$

と行われる．既存研究では，この計算を独立計算せず，節 5.1.1 のように攻撃した．つまり，(Q_{33}, m_{24}) を用いて CF_A を計算し，(Q_{17}, m_5) を用いて CF_B を計算し，後に式の一致を確かめた．

吸収特性

吸収特性はビットごとの論理関数の特性であり，この関数の入力に制限をかけることで，出力をコントロールできるというものである．

例えば，ビットごとの論理関数 $F(x,y,z) = (x \vee y) \wedge (\neg x \vee z)$ の吸収特性としては以下がある．

特性 1　$x = 1$ のとき，$F(x,y,z) = y$ となり，z に依存しない出力になる．

特性 1　$x = 0$ のとき，$F(x,y,z) = z$ となり，y に依存しない出力になる．

特性 2　$y = z = t$ のとき，$F(x,y,z) = t$ となり，x に依存しない出力になる．

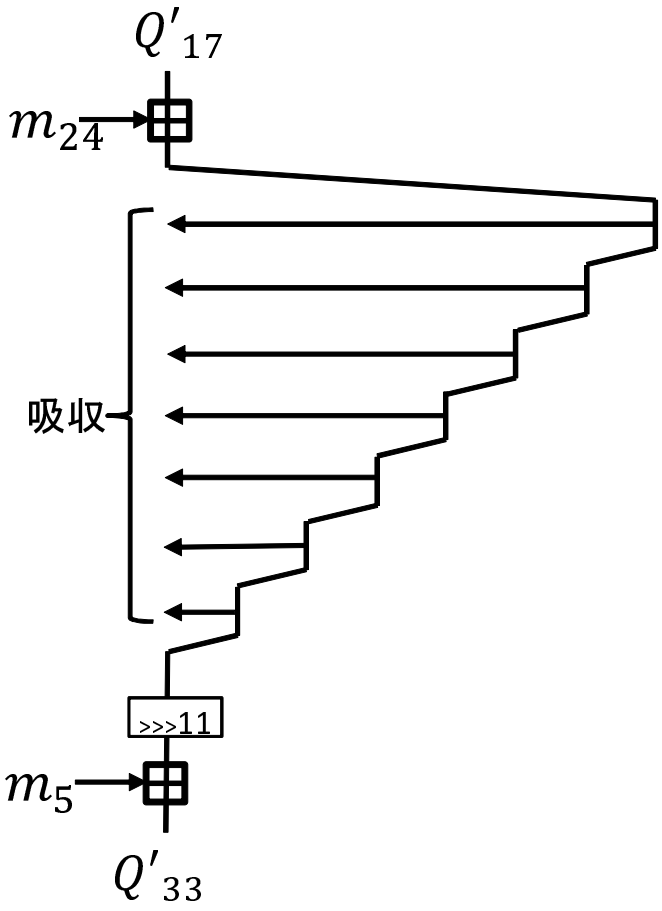

図 5.6：既存研究の IS(4-passHAVAL への擬似原像攻撃)

5.4.3　既存攻撃の計算量・メモリ量

(Q_{33}, m_{24}) 及び (Q_{17}, m_5) はどちらも 64 ビットを変数として動かせる．節 5.1.1 において $r = 32$ とすると，計算量 2^{224}, メモリ量 2^{64}words で擬似原像が求まる．

5.4.4 提案攻撃

提案攻撃では，図 5.7 のように既存研究では独立計算できなかった Q'_{17} から Q'_{33} までの計算を，図 5.7 のように独立計算可能な計算に置き換えることで，メモリ量を減らした．しかし，変換後の計算と元の計算が等価になるには条件が必要であり，条件を満たすようにしたところ計算量が増加した．この変更以外 (関数分け，PM,IS の吸収特性の使用方法など) は既存研究を踏襲している．

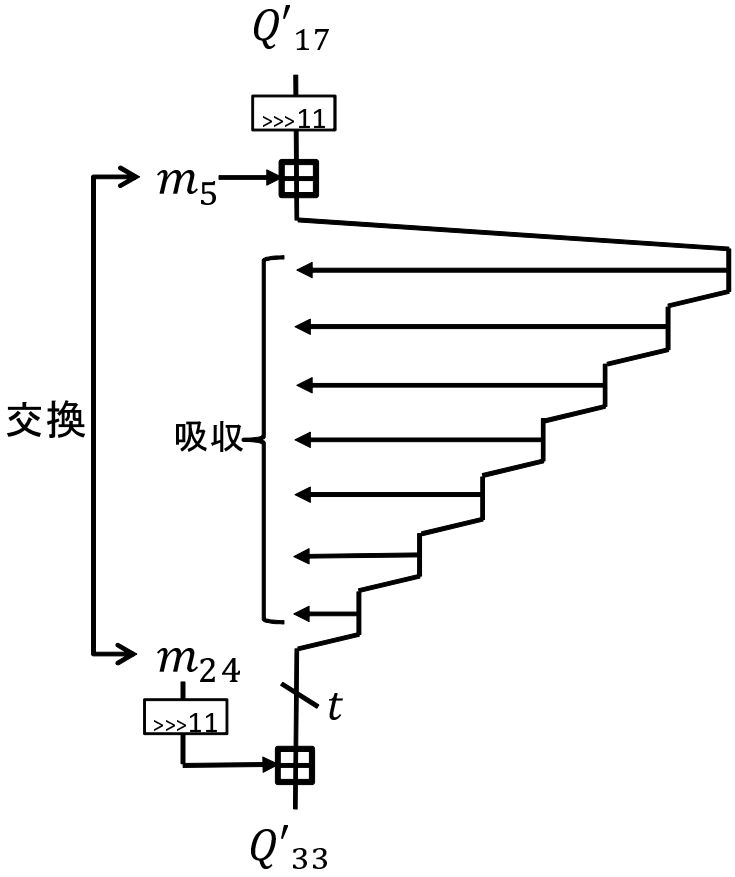

図 5.7 : 提案攻撃の IS(4-passHAVAL への擬似原像攻撃)

5.4.5 提案攻撃の IS

Q'_{17} から Q'_{33} までの計算を独立計算可能な等価な式に置き換える．

元の式は

$$(Q'_{17} + m_{24})^{\gg 11} + m_5 = Q'_{33} \tag{5.5}$$

である．

$$(Q'_{17} + m_{24})^{\gg 11} = {Q'_{17}}^{\gg 11} + m_{24}^{\gg 11} \tag{5.6}$$

のとき，式 (5.6) は

$${Q'_{17}}^{\gg 11} + m_5 = Q'_{33} - m_{24}^{\gg 11} \tag{5.7}$$

と，等価な計算に変換でき，

$$Q'_{17}{}^{\gg 11} + m_5 = Q'_{33} - m_{24}^{\gg 11} = t \tag{5.8}$$

として t を固定すると，

$$Q'_{17} = (t - m_5)^{\ll 11} \tag{5.9}$$

と Q'_{17} は m_5 の値によって一意に決まるようになり，また，

$$Q'_{33} = t + m_{24}^{\gg 11} \tag{5.10}$$

と，Q_{17} も m_{14} の値によって一意に決まるようになる．

結局，式 (5.6) が成り立つかどうかで等価な計算に変換できるかどうかが決まる．式 (5.6) が成り立つ条件は，以下 1. 及び 2. がどちらも成り立つことである．

1. $Q'_{17} + m_{24}$ で 31 から 32 ビットへの繰り上がりの切り捨てが発生しない (桁溢れが発生しない)
2. $Q'_{17}{}^{\gg 11} + m_{24}^{\gg 11}$ の計算で 31 から 32 ビットへの繰り上がりが発生しない (桁溢れが発生しない)

1. が成り立たない場合，つまり，桁溢れが発生する場合，式 (5.6) の左辺で桁溢れが反映されない．また，2. が成り立たない場合，つまり，桁溢れが発生する場合，式 (5.6) の左辺で桁溢れが反映されない．

本研究では 1. 及び 2. が常に成り立つようにするため，Q'_{17} 及び m_{24} のそれぞれの 31 ビット目と 10 ビット目の値を 0 に固定した．このとき式 (5.6) が必ず成り立つことをプログラムで確認した．攻撃の際は $m_5 = t - Q'_{17}{}^{\gg 11}$ で計算した m_5 を用いて攻撃する．

5.4.6 提案攻撃の計算量・メモリ量

提案攻撃では m_5 及び m_{24} をそれぞれ 30 ビットの変数として攻撃する．節 3.2 で $r = 30$ とすると，擬似原像攻撃の計算量は 2^{226}, メモリ量は 2^{30}words となる．また，原像攻撃は $2^{242}(= 2 * 2^{226+256}/2)$ となる．

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$\pi(j)$	0	1	2	3	4	5	⑥	7	8	9	10	11	12	13	⑭	15
							CF_B								Initial	
j	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
$\pi(j)$	1	⑥	11	0	5	10	15	4	9	⑭	3	8	13	2	7	12
	Structure							CF_A								
j	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
$\pi(j)$	5	8	11	⑭	1	4	7	10	13	0	3	⑥	9	12	15	2
					CF_A							skip				
j	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
$\pi(j)$	0	7	⑭	5	12	3	10	1	8	15	⑥	13	4	11	2	9
	skip			CF_B												

表 5.1：MD5 の関数分け

j	0	1	2	3	4	5	6	7	⋯	22	23	24	25	26	27	28	29	30	31	
$\pi(j)$	0	1	2	3	4	⑤	6	7	⋯	22	23	㉔	25	26	27	28	29	30	31	
							CF_B							Initial Structure						
j	32	33	⋯	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	
$\pi(j)$	⑤	14	⋯	30	3	21	9	17	㉔	29	6	19	12	15	13	2	25	31	27	
												CF_A								
j	64	65	66	67	68	69	70	71	72	73	74	75	76	77	⋯	92	93	94	95	
$\pi(j)$	19	9	4	20	28	17	8	22	29	14	25	12	㉔	30	⋯	23	11	⑤	2	
							CF_A											skip		
j	96	97	98	⋯	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127	
$\pi(j)$	㉔	4	0	⋯	11	31	21	8	27	12	9	1	29	⑤	15	17	10	16	13	
											CF_B									

表 5.2：4-passHAVAL の関数分け

第6章

まとめと考察

IS を改良したことにより，SHA-0 及び SHA-1 への攻撃ステップ数を増やした．また，MD5 及び 4-passHAVAL への攻撃に必要なメモリ量を激減させることができた．

成果 1 で用いたビット単位の IS は，構築する際に難しい条件を必要としないため，他の様々な中間一致攻撃に適用できると考えられる．

また，成果 2 は実際に使用されているハッシュ関数に対する攻撃を劇的に改良したため，研究のインパクトは非常に強いと考えられる．

第7章

謝辞

太田和夫教授，崎山一男准教授，岩本貢特任准教授，NTT 情報プラットフォーム研究所の佐々木悠様，南洋大学の王磊研究員には研究内容関して様々な助言をいただき，また，研究の進め方に関しても懇切丁寧に指導していただきました．深く感謝いたします．

また，太田・岩本研究室および崎山研究室の学生とは研究に関して様々な意見交換を行い，研究を行う上で非常に大きな助けとなりました．多大に感謝いたします．

参 考 文 献

[1] Yu Sasaki and Kazumaro Aoki. Finding Preimages in Full MD5 Faster Than Exhaustive Search. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 134–152. Springer, 2009.

[2] Yu Sasaki and Kazumaro Aoki. Preimage Attacks on 3, 4, and 5-Pass HAVAL. In *ASIACRYPT*, volume 5350 of *LNCS*, pages 253–271. Springer, 2008.

[3] Kazumaro Aoki and Yu Sasaki. Preimage Attacks on One-Block MD4, 63-Step MD5 and More. In *Selected Areas in Cryptography*, volume 5381 of *LNCS*, pages 103–119. Springer, 2008.

[4] National Institute of Standards and Technology. Secure Hash Standard (SHS)(Federal Information Processing Standards Publication 180-3), 2008.

[5] Kazumaro Aoki and Yu Sasaki. Meet-in-the-Middle Preimage Attacks Against Reduced SHA-0 and SHA-1. In *CRYPTO*, volume 5677 of *LNCS*, pages 70–89. Springer, 2009.

[6] Ronald L. Rivest. Request for Comments (RFC 1321): The MD5 Message-Digest Algorithm. The Internet Engineering Task Force, 1992.

[7] Yuliang Zheng, Josef Pieprzyk, and Jennifer Seberry. HAVAL — a one-way hashing algorithm with variable length of output. In *AUSCRYPT*, volume 718 of *LNCS*, pages 83–104. Springer, 1992.

[8] Alfred J.Menezes, Paul C. van Oorschot, and Scott A. Vanstone. Handbook of Applied Cryptography. *CRC Press*, 1997.

[9] John Kelsey and Bruce Schneier. Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work. In *EUROCRYPT*, volume 3494 of *LNCS*, pages 474–490. Springer, 2005.