ハッシュ関数(SHA-1)の安全性評価および攻撃手法整理

2006年3月

概 要

本報告書は，Wang らにより提案されたハッシュ関数 SHA-1 への攻撃法をまとめ，その問題点を洗い出し，課題を整理することが第一の目的である．SHA-1 は，米国標準技術局 (NIST) により，1995 年に米国政府標準方式として採用されたハッシュ関数である．SHA-1 は，その前のバージョンである SHA-0 におけるメッセージ拡大に少しの変形を加えることにより，提案されている．その提案以後，長い間，攻撃は提案されてこなかったが，2005 年になり，いくつかの提案がなされている．その結果，攻撃に必要な計算量として 2^{63} まで低下している．しかしながら，論文中には，詳しい記述がなされておらず，不明な点も多く，その有効性に関しては，詳細な検証が必要である．本年度は，Wang らの論文の精査を行ない，その問題点，課題点の整理を主眼として，調査活動を行なった．さらに，Wang らの論文を正確に理解するのを助かる目的で，さらに，Wang らの攻撃のさらなる効率化をはかる目的で，それ以外のハッシュ関数 MD4, MD5, SHA-0 に対する攻撃の改良を行なっている．SHA-1 は、これらのハッシュ関数を基にしているため、この改良は極めて重要である。これが第二の目的である．具体的には，MD4, MD5, SHA-0 の message modification の適用範囲の拡大を行ない，実際に計算量の削減をはかっている．また，MD5 においては，差分パスから，sufficient condition の求める方法に関する研究を行ない，実際に，Wang らとは異なる sufficient condition を求めている．

本報告書の構成

本報告書の構成は以下の通りである．第 1 章で、SHA-1 の仕様をまとめ，SHA-1 に対する既存の攻撃を簡単に紹介している．ついで，第 2 章から、Wang らの論文をもとに，その攻撃の詳細を述べ，問題点，不明な点を明らかにしている．

Wang の SHA-1 への攻撃法を理解する上で，MD4, MD5, SHA-0 などの SHA-1 よりも構造が簡単なハッシュ関数に対する攻撃を理解することは極めて重要である．さらに，Wang らの成果をさらに効率的なものにするためには，これらのハッシュ関数に対する攻撃法の改良を検討することは極めて重要である．その目的のために，各方面からの検討を行なった．例えば，message modification の適用範囲の拡大を行なう．具体的には，MD4, MD5、SHA-0 に対して行なっている．さらに，差分パスから sufficient condition の探求を MD5 を対象に行なっている．本報告書では，5 章から 8 章にその結果をまとめて掲載している．具体的には，5 章，6 章で，それぞれ MD4，MD5 に関する message modification の改良法を提案している．また，7 章では，MD5 において，差分パスが与えられた時の sufficient condition の求め方を提案している．8 章では，SHA-0 に対する message modification を提案している．9 章では，本報告書のまとめを行ない，今後の研究課題を示す．

目 次

第1章　SHA-1の仕様・攻撃の歴史

本章ではハッシュ関数SHA-1の仕様と，これまでに提案されたSHA-1への衝突攻撃を紹介する．

1.1　SHA-1の仕様

SHA-1は1995年に米国標準技術局によって発表されたハッシュ関数である．SHA-1は長さlビット$(0 \leq l \leq 2^{64})$のメッセージから160ビットのハッシュ値を生成する．1.1章ではハッシュ値の計算方法を説明する．

1.1.1　パディング

まずはじめに入力メッセージMをパディングし，メッセージ長を圧縮関数の入力長である512ビットの倍数にそろえる．パディングの手順を図1に示す．

図 1.1: SHA-1のパディング方法

パディングでは，Mの後ろに1を加え，その後，(次の512の倍数-64)ビット目まで0を加える．最後に入力メッセージ長を64ビットで表した

ものを加える．この操作により，M の長さは 512 の倍数になることが保証される．

次に M を n 個の 512 ビットのメッセージに分割する．分割された最初のメッセージを M_0，2番目のメッセージを M_1，$\cdots$，$N-1$ 番目のメッセージを M_N とする．

$$M = (M_0, M_1, \ldots M_N), \qquad |M_i| = 512$$

512 ビットのメッセージは，16 個の 32 ビットのメッセージで表される．M_i の最初の 32 ビットを $m_0^{(i)}$，次の 32 ビットを $m_1^{(i)}$，$\cdots$，16 番目の 32 ビットを $m_{15}^{(i)}$ と表す．

$$M_i = (m_0^{(i)}, m_1^{(i)}, \ldots m_{15}^{(i)}), \qquad |m_j^{(i)}| = 512$$

1.1.2 ハッシュ値の初期値

SHA-1 のハッシュ値の初期値 $H^{(0)}$ は次のように定義されている．

$$H_0^{(0)} = 0\text{x}67452301$$
$$H_1^{(0)} = 0\text{x}efcdab89$$
$$H_2^{(0)} = 0\text{x}98badcfe$$
$$H_3^{(0)} = 0\text{x}10325476$$
$$H_4^{(0)} = 0\text{x}c3d2e1f0$$

1.1.3 ハッシュ値の計算

各メッセージブロック $M^{(i)}$ を使い，次のようにハッシュ値を計算する．

For $i = 1$ to N
{

1. メッセージ拡大により $\{W_t\}$ を計算する．

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

ここで，$ROTL^k$ は左 k ビット巡回シフトである．

2. 内部変数 a, b, c, d, e を $(i-1)$ 番目のハッシュ値に初期化する.

$$
\begin{aligned}
a &= H_0^{(i-1)} \\
b &= H_1^{(i-1)} \\
c &= H_2^{(i-1)} \\
d &= H_3^{(i-1)} \\
e &= H_4^{(i-1)}
\end{aligned}
$$

3. For i=0 to 79
   {

$$
\begin{aligned}
T &= ROTL^5(a) + f_t(b,c,d) + e + K_t + W_t \\
e &= d \\
d &= c \\
c &= ROTL^{30}(b) \\
b &= a \\
a &= T
\end{aligned}
$$

   }

   ここで，論理関数 f_t と定数 K_t は次の通り.

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

$$
k_t = \begin{cases}
\text{0x}5a827999 & 0 \le t \le 19 \\
\text{0x}6ed9eba1 & 20 \le t \le 39 \\
\text{0x}8f1bbcdc & 40 \le t \le 59 \\
\text{0x}ca62c1d6 & 60 \le t \le 79
\end{cases}
$$

   内部変数の更新過程を図にすると次のようになる.

4. i番目の中間ハッシュ値$H^{(i)}$を計算する．

$$
\begin{aligned}
H_0^{(i)} &= a + H_0^{(i-1)} \\
H_1^{(i)} &= b + H_1^{(i-1)} \\
H_2^{(i)} &= c + H_2^{(i-1)} \\
H_3^{(i)} &= d + H_3^{(i-1)} \\
H_4^{(i)} &= e + H_4^{(i-1)}
\end{aligned}
$$

}

最終的な入力メッセージMのハッシュ値は次のようになる．

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

1.2 SHA-1に対する衝突攻撃の歴史

SHA-1に対する主な衝突攻撃の論文としては，Eurocrypto '05 でEli Biham, Rafi Chen, Antoine Joux, Patrick Carribault,Christophe Lemuet and William Jalbyによって発表された40stepまでのコリジョン攻撃が挙げられる．彼らはこの攻撃の中で，マルチブロックテクニックというものを新たに導入している．

また同時期に，Xiaoyun Wang, Yiqun Lisa Yin, Hongbo Yuらにより SHA-1のコリジョンは2^{69}のSHA-1演算以下で発見できるという発表があった．また，SHA-1の58ステップまでのコリジョンが発表された．この報告には，彼女らのグループが発見した結果を載せているだけで，技術的な報告は一切なかった．

次に，CRYPTO'05で同グループが，SHA-1のコリジョンを計算量2^{69}のSHA-1演算以下で発見できる方法の詳細を発表した．また，同会議rump sessionおよび，NIST HASH WORKSHOP 2005において，この攻撃の改善版である計算量2^{63}のSHA-1演算以下でコリジョンを発見する方法が発表された．

図 1.2: SHA-1 の内部変数の更新方法

第2章　Wangらの方式の説明

本章でWangらが提案した，SHA-1に対するコリジョンアタック方式について説明する．Wangらの方式は減算を差分として用いる，差分攻撃法である．コリジョン探索に必要な計算量は2^{69}である．

Wangらの方式でコリジョンを発見するためには，次の2つの作業が必要になる．

1. 差分パスを生成する．
2. 1で求めた差分パスからコリジョン探索を行う．

2.1　記号，用語の説明

2.1.1　用語の説明

- 差分: 引き算を意味する．
- 差分値: ある2変数を引き算した値を意味する．
- 内部変数: 更新変数a, b, c, d, eを意味する．
- ステップ: 内部変数更新1回分の計算．
- iステップ目: i回目の内部変数更新の計算
- 1ラウンド目: ステップ1からステップ20を意味する．
- 2ラウンド目: ステップ21からステップ40を意味する．
- 3ラウンド目: ステップ31からステップ60を意味する．
- 4ラウンド目: ステップ61からステップ80を意味する．

2.1.2 記号の説明

- a_i: i ステップ目計算終了後の内部変数 a の値.
- b_i: i ステップ目計算終了後の内部変数 b の値.
- c_i: i ステップ目計算終了後の内部変数 c の値.
- d_i: i ステップ目計算終了後の内部変数 d の値.
- e_i: i ステップ目計算終了後の内部変数 e の値.
- m_i: i ステップ目計算終了後の計算で用いられるメッセージ．1 章の W_i と同じ意味を持つ記号である．
- $\lll j$: j ビット左巡回シフト．1 章の $ROT^j()$ と同じ意味を持つ記号である．
- $f()$: 各ラウンドごとに使用が定められている論理関数．1 章では i ステップ目の論理関数を $f_i()$ と記述していたが，2 章では添え字を記述せずに話を進める．
- Δx: 変数 x の差分値.
- $x_{i,j}$: ステップ i の変数 x の j ビット目の値.

2.2 差分パス生成の方法

本章では，差分パスを生成する方法について説明する．差分パスを生成するための最も重要な部品の 1 つにローカルコリジョンがある．これは，任意のステップで任意のビットに差分を入力しその差分を 6 ステップで打ち消す方式である．このローカルコリジョンを SHA-1 に当てはめて，差分パスを生成する．次に，ローカルコリジョンを発生させる位置を決定するためにディスターバンスベクトルというローカルコリジョンのスタート位置を示すベクトルを用意する．このベクトルを用いてローカルコリジョンの発生位置を決定する．ローカルコリジョンの発生位置が決定されると，メッセージ差分 ΔM と内部変数の差分が決定される．これらの差分を実現するための十分条件として sufficient condition と呼ばれる条件を求める．以上が差分パス生成方法の概要である．次に，ローカ

ルコリジョン，ディスターバンスベクトル，sufficient condition について説明をする．

2.3 ローカルコリジョン

ローカルコリジョンは，表 1 のように任意のステップ i で任意のビット $(j+1$ ビット目$)$ に差分を入力しその差分を 6 ステップで打ち消す方式である．

step	Δm	Δa	Δb	Δc	Δd	Δe
i	$\pm 2^j$	$\pm 2^j$				
$i+1$	$\pm 2^{j+5}$		$\pm 2^j$			
$i+2$	$\pm 2^j$			$\pm 2^{j-2}$		
$i+3$	$\pm 2^{j-2}$				$\pm 2^{j-2}$	
$i+4$	$\pm 2^{j-2}$					$\pm 2^{j-2}$
$i+5$	$\pm 2^{j-2}$					

表 2.1: ローカルコリジョン

次にローカルコリジョンのステップごとの処理について詳しく説明する．

ステップ i

まずステップ i で差分 $\Delta m_{i-1} = \pm 2^j$ を入力する．この差分に対する桁上がりはないものとして扱う．a_i は以下のように計算されるので，$\Delta m_{i-1} = \pm 2^j$ によって $\Delta a_{i-1} = \pm 2^j$ となる．

$$a_i = (a_{i-1} \lll 5) + f(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + m_{i-1} + k_{i-1}$$

ステップ $i+1$

6 ステップで差分を打ち消すためには $\Delta a_{i+1} = 0$ でなければならない．ステップ $i+1$ で a_{i+1} は以下のように計算されるので，$\Delta a_{i+1} = 0$ となるように，$\Delta a_i = \pm 2^j$ を $\Delta m_i = \pm 2^{j+5}$ で打ち消す．ここで，Δa_i の符号の向きと Δm_i の符号の向きは逆になっている．

$$a_{i+1} = (a_i \lll 5) + f(b_i, c_i, d_i) + e_i + m_i + k_i$$

ステップ $i+2$

6 ステップで差分を打ち消すためには $\Delta a_{i+2} = 0$ でなければならない．ここで，$\Delta b_{i+1} = 2^j$ であるが，以下のように b_{i+1} は関数 f の計算で用いられるので，関数 f の差分について考えなければならない．

$$a_{i+2} = (a_{i+1} \lll 5) + f(b_{i+1}, c_{i+1}, d_{i+1}) + e_{i+1} + m_{i+1} + k_{i+1}$$

ローカルコリジョンはどのステップでも実現可能でなければならないので，$\Delta f(b_{i+1}, c_{i+1}, d_{i+1}) = \pm 2^j$ となる．ここで，関数 f の差分の符号の向きは関数 f によって異なってくる．この関数 f の差分を $\Delta m_{i+1} = \pm 2^j$ で打ち消す．ここで，$\Delta f(b_{i+1}, c_{i+1}, d_{i+1})$ の符号の向きと Δm_{i+1} の符号の向きは逆になる．

ステップ $i+3$

6 ステップで差分を打ち消すためには $\Delta a_{i+3} = 0$ でなければならない．ここで，$\Delta c_{i+2} = 2^{j-2}$ であるが，以下のように c_{i+2} は関数 f の計算で用いられるので，関数 f の差分について考えなければならない．

$$a_{i+3} = (a_{i+2} \lll 5) + f(b_{i+2}, c_{i+2}, d_{i+2}) + e_{i+2} + m_{i+2} + k_{i+2}$$

ローカルコリジョンはどのステップでも実現可能でなければならないので，$\Delta f(b_{i+2}, c_{i+2}, d_{i+2}) = \pm 2^{j-2}$ となる．ここで，関数 f の差分の符号の向きは関数 f によって異なってくる．この関数 f の差分を $\Delta m_{i+2} = \pm 2^{j-2}$ で打ち消す．ここで，$\Delta f(b_{i+2}, c_{i+2}, d_{i+2})$ の符号の向きと Δm_{i+2} の符号の向きは逆になる．

ステップ $i+4$

6 ステップで差分を打ち消すためには $\Delta a_{i+4} = 0$ でなければならない．ここで，$\Delta d_{i+3} = 2^{j-2}$ であるが，以下のように d_{i+3} は関数 f の計算で用いられるので，関数 f の差分について考えなければならない．

$$a_{i+4} = (a_{i+3} \lll 5) + f(b_{i+3}, c_{i+3}, d_{i+3}) + e_{i+3} + m_{i+3} + k_{i+3}$$

ローカルコリジョンはどのステップでも実現可能でなければならないので，$\Delta f(b_{i+3}, c_{i+3}, d_{i+3}) = \pm 2^{j-2}$ となる．ここで，関数 f の差分の符号の

向きは関数 f によって異なってくる．この関数 f の差分を $\Delta m_{i+3} = \pm 2^{j-2}$ で打ち消す．ここで，$\Delta f(b_{i+3}, c_{i+3}, d_{i+3})$ の符号の向きと Δm_{i+3} の符号の向きは逆になる．

ステップ $i+5$

6 ステップで差分を打ち消すためには $\Delta a_{i+5} = 0$ でなければならない．ここで，$\Delta e_{i+4} = 2^{j-2}$ であるが，以下のように e_{i+4} は関数 f の計算で用いられるので，関数 f の差分について考えなければならない．

$$a_{i+5} = (a_{i+4} \lll 5) + f(b_{i+4}, c_{i+4}, d_{i+4}) + e_{i+4} + m_{i+4} + k_{i+4}$$

ローカルコリジョンはどのステップでも実現可能でなければならないので，$\Delta f(b_{i+4}, c_{i+4}, d_{i+4}) = \pm 2^{j-2}$ となる．ここで，関数 f の差分の符号の向きは関数 f によって異なってくる．この関数 f の差分を $\Delta m_{i+4} = \pm 2^{j-2}$ で打ち消す．ここで，$\Delta f(b_{i+4}, c_{i+4}, d_{i+4})$ の符号の向きと Δm_{i+4} の符号の向きは逆になる．

2.4　ディスターバンスベクトル

本章では，ディスターバンスベクトルについて説明する．

ディスターバンスベクトルはローカルコリジョンのスタート位置を示すベクトルである．ディスターバンスベクトル $x_{-5}, ..., x_{-1}, x_0, ..., x_{79}$ を用意する．x_i $(i = 0, ..., 79)$ は 32 ビットである．このディスターバンスベクトルを以下のように定義する．

- ステップ i からローカルコリジョンがスタートする場合: $x_{i-1} = 1$
- ステップ i からローカルコリジョンがスタートしない場合: $x_{i-1} = 0$

ディスターバンスベクトルはローカルコリジョンの最初のメッセージ差分が入力される位置を示しているので，メッセージ拡大と同様に $x_{16} - x_{79}$ を以下のように計算する．

$$x_i = (x_{i-3} \oplus x_{i-8} \oplus x_{i-14} \oplus x_{i-16}) \lll 1$$

また，ステップ $11 - 16$ からスタートするローカルコリジョンが期待通りに動作する条件として，ディスターバンスベクトルに以下の条件が必要になる．

- $x_{11} = (x_{-5} \oplus x_{-3} \oplus x_3 \oplus x_8) \lll 1$
- $x_{12} = (x_{-4} \oplus x_{-2} \oplus x_4 \oplus x_9) \lll 1$
- $x_{13} = (x_{-3} \oplus x_{-1} \oplus x_5 \oplus x_{10}) \lll 1$
- $x_{14} = (x_{-2} \oplus x_0 \oplus x_6 \oplus x_{11}) \lll 1$
- $x_{15} = (x_{-1} \oplus x_1 \oplus x_7 \oplus x_{12}) \lll 1$

以上の条件の下でディスターバンスベクトルを探索するが，探索空間は 2^{512} と非常に大きくなるので，全数探索を行い最適なものを選択するのは困難である．そこで，Wang らはヒューリスティックに探索を行い，効率がよさそうなものを選択している．この探索方法は sufficient condition の後で説明する．

2.5 Sufficient Condition

ディスターバンスベクトルの値が決定すると，メッセージと内部変数の差分が決定する．しかし，この差分をコントロールするために条件が必要になる．この条件が sufficient condition である．sufficient condition はそれぞれのラウンドごとに条件が異なる．ラウンドごとの条件は以下の通りである．ただし，この条件は $x_{i,j} = 1$ かつ $x_{i,j+1} = 0$ かつ $x_{i,j-1} = 0$ の時の条件である．

- $x_{i,j} = 1$ のとき．(ただし，2 ラウンド目と 4 ラウンド目)

$$a_{i-1,j+2} = a_{i-2,j+2} \ (or \ a_{i-1,j+2} \stackrel{\prime}{\neq} a_{i-2,j+2}) \tag{2.1}$$
$$a_{i,j} = m_{i-1,j} \tag{2.2}$$
$$a_{i+1,j-2} = a_{i-1,j} \ (or \ a_{i+1,j-2} \stackrel{\prime}{\neq} a_{i-1,j}) \tag{2.3}$$
$$a_{i+2,j-2} = a_{i+1,j} \ (or \ a_{i+2,j-2} \stackrel{\prime}{\neq} a_{i+1,j}) \tag{2.4}$$
$$m_{i-1,j} \stackrel{\prime}{\neq} m_{i+1,j+5} \tag{2.5}$$

ただし，$j = 2$ の時は条件 (3)，(4) を無視することができる．

- $x_{i,j} = 1$ のとき．(ただし，3 ラウンド目)

$$a_{i-1,j+2} \neq a_{i-2,j+2} \tag{2.6}$$
$$a_{i,j} = m_{i-1,j} \tag{2.7}$$
$$a_{i+1,j-2} \neq a_{i-1,j} \tag{2.8}$$
$$a_{i+2,j-2} \neq a_{i+1,j} \tag{2.9}$$
$$m_{i-1,j} \neq m_{i,j+5} \tag{2.10}$$
$$m_{i-1,j} \neq m_{i+1,j} \tag{2.11}$$
$$m_{i-1,j} \neq m_{i+2,j-2} \tag{2.12}$$
$$m_{i-1,j} \neq m_{i+3,j-2} \tag{2.13}$$
$$m_{i-1,j} \neq m_{i+4,j-2} \tag{2.14}$$

ただし，$j = 2$ のときは条件 (12)，(13)，(14) を無視することができる．

ディスターバンスベクトルの中に 1 が連続して並んでいた場合条件を減らすことができる．例として，$x_{i,1} = 1, x_{i,2} = 1$ の場合について説明する．この場合，ステップ i で $\Delta m_{i-1} = \pm 2^1 \pm 2^0$ の差分が入力される．この差分について，2^1 の符号の向きが “+” の場合 2^0 の符号の向きを “−” にすると，$\Delta m_{i-1} = 2^1 - 2^0 = 2^0$，$2^1$ の符号の向きが “+” の場合 2^0 の符号の向きを “−” にすると $\Delta m_{i-1} = -2^1 + 2^0 = -2^0$ となり，2 つ分の差分を 1 つにすることができる．ステップ $i+1, i+2.i+3, i+4, i+5$ に関しても同様に 2 つの差分を 1 つにすることができる．このようにすると，ローカルコリジョン 2 つ分を条件 1 つ分にすることができる．

2.6 ディスターバンスベクトルの探索法

ディスターバンスベクトルの探索空間は 2^{512} と非常に広く，全数探索は困難である．そこで，Wang らはヒューリスティックな方法で，探索を行いそのディスターバンスベクトルがベストなものかどうかを検証することは難しいが，効率のよさそうなディスターバンスベクトルを求めている．本章ではその方法について説明する．

まず，よいディスターバンスベクトルを探索するためには以下のことに注意し探索を行う．

1. ローカルコリジョンのスタート位置が2ビット目のときは，他のビットからスタートするよりも条件を減らすことができる．

2. ディスターバンスベクトルで1が隣り合って出現した場合，2つ分の条件を1つに減らすことができる．

3. 3ラウンド目の条件は2ラウンド目，4ラウンド目の条件よりも多く必要である．

また，後で記述をするが，コリジョン探索をする際に，ステップ22までのsufficient conditionを確率1で満たす方式としてmessage modificationがある．この方式を考慮に入れると，ステップ$1-22$までのsufficient conditionの個数は計算量に影響しないので，ステップ23以降のsufficient conditionの個数が少ないディスターバンスベクトルを選択すればよい．このことと，上記の条件1，2を考慮に入れると，$x_{i,2}, ..., x_{i+15,2}, x_{i,1}, ..., x_{i+15,1}$ $(i = 0, ..., 64)$のみを探索すれば，ステップ23以降のsufficient conditionの個数が少ないディスターバンスベクトルを発見することができる可能性がある．ここで，$x_{i,3}, ..., x_{i,32}, ..., x_{i+15,3}, ..., x_{i+15,32}$は全て0と設定しておく．これにより，ディスターバンスベクトルの探索空間を2^{512}から$64 \times 2^{32} = 2^{38}$に減少するので，この範囲での全数探索が可能になる．さらに，上記の条件3を考慮に入れると，この探索範囲で最もよいディスターバンスベクトルとして，表2のディスターバンスベクトルが候補として挙げられる．

表2のディスターバンスベクトルはWangらが選択したものである．このディスターバンスベクトルからsufficient conditionを求めると，ステップ23以降のsufficient conditionの個数は71個になる．

2.7 1ラウンド目のパス生成法

1ラウンド目の算術差分の決め方は2-4ラウンド目の算術差分の決め方とは異なる．2-4ラウンド目はローカルコリジョンにしたがってパスを生成すればいいが，1ラウンド目は関数fの特徴やディスターバンスベクトルに$x_{-5}, ..., x_{-1}$があるため，1ラウンド目はローカルコリジョンのルール通りには算術差分を生成することができない．そのため，1ラウンド目の算術差分の決め方は2ラウンド目以降でつじつまが合うような算術差分を立てる．このつじつまの合わせ方は1ラウンド目の差分による桁上がりと関数fの特徴を組みあわせることにより可能である．

2.8　2ブロックコリジョンパス

表2のディスターバンスベクトルを基にメッセージを探索すると，$x_{76} = 40, x_{78} = 28, x_{79} = 80$ となっているのでコリジョンを求めることができず，ニアーコリジョンを求めることになる．そのためこのディスターバンスベクトルから1ブロックでコリジョンを発見することは不可能である．そこで，Wangらは2ブロックを用いたコリジョンパス方式を提案している．このパスから1ブロック目の出力差分は以下の通りになる．

$$(\Delta aa1_{80}, \Delta bb1_{80}, \Delta cc1_{80}, \Delta dd1_{80}, \Delta ee1_{80}) = (\pm 2^7, \pm 2^4 \pm 2^3, 0, \pm 2^3, 0)$$

2ブロック目に関して，1ブロック目と同じディスターバンスベクトルを用いると2ブロック目計算終了後の内部変数の差分値は以下のようになる．

$$(\Delta a2_{80}, \Delta b2_{80}, \Delta c2_{80}, \Delta d2_{80}, \Delta e2_{80}) = (\pm 2^7, \pm 2^4 \pm 2^3, 0, \pm 2^3, 0)$$

2ブロック目の出力は $(aa_1 + a_2, bb_1 + b_2, cc_1 + c_2, dd_1 + d_2, ee_1 + e_2)$ なので，2ブロック目の内部変数の差分の符号の向きを1ブロック目の出力差分の向きとは逆にすれば2ブロック目の出力差分を0にすることができ，2ブロックでコリジョンを起こすことができる．符号を逆にするために，sufficient conditionを設定すれば実現することができる．

2.9　コリジョン探索

本章ではコリジョン探索について説明する．コリジョン探索は差分パス生成時に求めたsufficient conditionを全て満たすメッセージを探索する．探索手順は以下の通りである．ここで，以下の手順で用いられているmessage modificationはステップ1からステップ22のsufficient conditionを満たすメッセージを効率的に求めることができる手法である．message modificationの説明は後で説明する．

1. message modificationを用いて1ブロック目のステップ1からステップ14のsufficient conditionを満たす内部変数を求めることができるメッセージ $m_0, ..., m_{13}$ を求める．

2. message modificationを用いて1ブロック目のステップ15，ステップ16のsufficient conditionを満たす内部変数を求めることができるメッセージ m_{14}, m_{15} を求める．

3. message modification を用いて 1 ブロック目のステップ 17 からステップ 22 の sufficient condition を満たす内部変数を求めることができるメッセージ $m_0, ..., m_{15}$ を求める．

4. 手順 1，2，3 で求めたメッセージ M_1 からステップ 23 以降の sufficient condition を満たす内部変数を計算できるかどうかチェックする．満たさなければ手順 2 に戻る．満たしていれば手順 5 に進む．

5. M_1 を用いて，2 ブロック目の差分パスを生成する．

6. message modification を用いて 2 ブロック目のステップ 1 からステップ 14 の sufficient condition を満たす内部変数を求めることができるメッセージ $m_0, ..., m_{13}$ を求める．

7. message modification を用いて 2 ブロック目のステップ 15，ステップ 16 の sufficient condition を満たす内部変数を求めることができるメッセージ m_{14}, m_{15} を求める．

8. message modification を用いて 2 ブロック目のステップ 17 からステップ 22 の sufficient condition を満たす内部変数を求めることができるメッセージ $m_0, ..., m_{15}$ を求める．

9. 手順 6，7，8 で求めたメッセージ M_2 からステップ 23 以降の sufficient condition を満たす内部変数を計算できるかどうかチェックする．満たさなければ手順 7 に戻る．満たしていれば手順 10 に進む．

10. 手順 1 から手順 9 で求めた sufficient condition を全て満たす内部変数を計算できるメッセージ M とメッセージ差分 ΔM を用いて，メッセージ M' を以下のように計算する．

$$M' = M + \Delta M$$

M, M' がコリジョンメッセージになる．

2.10 Message Modification

message modification はステップ 1 からステップ 22 の sufficient condition を満たすメッセージを効率的に求めることができる手法である．ステップ 1 からステップ 16 で用いられるメッセージは自由に設定でき，ステッ

プ 17 以降のメッセージはメッセージ拡大により決定され自由に設定できない．そのため，ステップ 1 からステップ 16 の message modification とステップ 17 以降の message modification は異なる方式になる．

ステップ i $(i = 1, ..., 16)$ の message modification

ステップ i $(i = 1, ..., 16)$ の message modification の手順は以下の通りである．

- ステップ i の sufficient condition を満たす内部変数 a_i を求める．
- $m_{i-1} = a_i - (a_{i-1} \lll 5) - f(b_{i-1}, c_{i-1}, d_{i-1}) - e_{i-1} - k_{i-1}$ を計算する．

ステップ i $(i = 17, ..., 22)$ の message modification

ステップ 17 からステップ 22 の message modification は sufficient condition を満たすように m_{10} から m_{15} を修正する．

index i	x_i	index i	x_i
−5	80000000	39	00000000
−4	00000002	40	00000000
−3	00000000	41	00000000
−2	80000001	42	00000002
−1	00000000	43	00000000
0	40000001	44	00000002
1	00000002	45	00000000
2	00000002	46	00000002
3	80000002	47	00000000
4	00000001	48	00000002
5	00000000	49	00000000
6	80000001	50	00000000
7	00000002	51	00000000
8	00000002	52	00000000
9	00000002	53	00000000
10	00000000	54	00000000
11	00000000	55	00000000
12	00000001	56	00000000
13	00000000	57	00000000
14	80000002	58	00000000
15	00000002	59	00000000
16	80000002	60	00000000
17	00000000	61	00000000
18	00000002	62	00000000
19	00000000	63	00000000
20	00000003	64	00000004
21	00000000	65	00000000
22	00000002	66	00000000
23	00000002	67	00000008
24	00000001	68	00000000
25	00000000	69	00000000
26	00000002	70	00000010
27	00000002	71	00000000
28	00000001	72	00000008
29	00000000	73	00000020
30	00000000	74	00000000
31	00000002	75	00000000
32	00000003	76	00000040
33	00000000	77	00000000
34	00000002	78	00000028
35	00000002	79	00000080
36	00000000		
37	00000000		
38	00000002		

表 2.2: ディスターバンスベクトル

第3章　攻撃の特徴的な技術をリストアップ

Wangらの攻撃法で最も特徴的な技術は算術演算を差分として用いたことである．秘密鍵への差分攻撃は排他的論理和を用いていたが，SHA-1は関数fと左巡回シフト以外の演算は全て32ビットの加算なので，算術演算を差分として用いた場合，例えばSHA-1のiステップ目のメッセージm_{i-1}に2^jの差分を入力した場合，そのステップで計算されるa_iに確率1で2^jの差分がたつ．つまり以下の等式が確率1で成り立つ．

$$a_i \pm 2^j = (a_{i-1} \lll 5) + f(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + m_{i-1} \pm 2^j + k_{i-1}$$

しかし，差分として排他的論理和を用いた場合，例えばSHA-1のiステップ目のメッセージm_{i-1}に2^jの差分を入力した場合，そのステップで計算されるa_iに排他的論理和の意味で2^jの差分がたつ確率は1ではなく$\frac{1}{2}$になる．つまり以下の等式が確率$\frac{1}{2}$で成り立つ．

$$a_i \oplus 2^j = (a_{i-1} \lll 5) + f(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + (m_{i-1} \oplus 2^j) + k_{i-1}$$

よって，算術差分を用いると上記のようにa_iへの差分の遷移が確率1で成り立つ．

次に，攻撃をするためにはこの算術差分を用いてSHA-1の内部変数にどのような算術差分をたてるかについて決めなければならない．2-4ラウンド目に関してはローカルコリジョンのルール通りに算術差分を決めればいいが，この技術はWangの論文以前からあった技術なのでこの論文の新規性には当たらない．では何が新しいかと言うと，最も計算量削減に役に立った技術として1ラウンド目の算術差分の決め方をローカルコリジョンのルール通りに作らずに別の方法で算術差分を決めたことである．これにより，Wangらの攻撃法で説明したディスターバンスベクトルに3つの条件がついていたのだが，そのうちの2つを取り除くことが可能になる．さらに，Wangらは2ブロックでのコリジョンアタック方式を提

案しているため，最後の 1 つの条件を取り除くことが可能になる．これにより，1 ラウンド目の 3 つの条件を全て取り除くことができ，これにより探索空間が広がりアタッカーにとって有利な状況を作ることができる．

内部変数の算術差分を決めた後，どんな SHA-1 への入力メッセージに対しても常に内部変数の差分が期待通りの算術差文になるとは限らない．例えば，以下の式で a_i に 2^6 の差分が必要な場合を考えてみる．

$$a_i = (a_{i-1} \lll 5) + f(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} + m_{i-1} + k_{i-1}$$

b_{i-1} にのみに 2^6 の差分がたち桁上がりが起きていない場合，この差分によって a_i の差分を立てる必要がある．b_{i-1} は差分による桁上がりが起きない状況を考えているので，b_{i-1} の 7 ビット目が 0 から 1 に変化する．しかし，関数 f の特徴から関数 f の差分が期待通りになるとは限らない．例えば，$f(b_{i-1}, c_{i-1}, d_{i-1}) = b_{i-1} \oplus c_{i-1} \oplus d_{i-1}$ の場合 2^6 の差分が現れる確率は $\frac{1}{2}$ になる．他の関数 f の場合も 2^6 の差分の発生は確率的である．そのため Wang らの方式では差分が期待通りに現れるための十分条件 (sufficient condition) を求め，この条件を満たすメッセージを探索している．上の例から求めることができる sufficient condition は，$f(b_{i-1}, c_{i-1}, d_{i-1}) = b_{i-1} \oplus c_{i-1} \oplus d_{i-1}$ の場合 $c_{i-1,7} = d_{i-1,7}$ である．また上の例では桁上がりが無い場合を考えていたが，この桁上がりが無い状況が起こるのも確率的である．そのため，Wang らは桁上がりについての sufficient condition ももとめている．この例の場合桁上がりが起きないための sufficient condition は $a_{i,7} = 0$ である．sufficient condition は上記で説明したとおり期待通りの算術差分が発生するための条件なので，sufficient condition を全て満たすメッセージを発見することができれば，コリジョンを発見することができるが，sufficinet condition の数はかなり多いので全ての sufficient condition を満たすメッセージを見つけることは困難である．さらに，sufficient condition を全て満たすメッセージを探索するのと全ての算術差分を満たすメッセージを探索するのは同じである．一見 sufficient condition は不必要に思うが，message modification という手法を組み合わせることにより非常に強力な効果を発揮する．message modification はステップ 1-22 の sufficient condition を効率的に満たす手法である．ステップ 1-16 の sufficient condition に対しては 16 ステップ分の計算量で満たすメッセージを発見することができる．しかし message modification を用いなかった場合，例えば 7 ステップ目の sufficient condition を満たすメッセージを発見するための計算量は 2^{22} ステップ掛かる．ステップ 17-22 に関しても，message modification を用いたほうがはるかに効率的に sufficient condition を満たすメッセージを発

見することができる．よって message modification を用いると，探索しなければならない sufficient condition の数は 71 個に減り，探索の計算量は 2^{69} になる．

以上が Wang らの攻撃法の特徴であるが，これらをまとめると以下のようになる．

- 差分として算術演算を用いる (Wang らの攻撃で最も重要な特徴)
- 1 ラウンド目の差分の決め方と 2 ブロックを用いた探索法を用いることによりディスターバンスベクトルの探索空間を広げたこと
- sufficient condition と message modification を組み合わせることにより，コリジョン探索を効率化したこと

Wang らはこの 3 つのことを組み合わせることで，SHA-1 の 2^{80} の壁を越えることができた．

第4章　Wangらの攻撃で不明な点

本章ではWangらの攻撃で不明な点についてあげる．Wangらの攻撃で不明な点は以下の4つである．

1. 1ブロック目の1ラウンド目の算術差分の決め方
2. 2ブロック目の1ラウンド目の算術差分の決め方
3. Wangらのディスターバンスベクトルの最適性
4. ステップ17以降のmessage modification

これらの点について，なぜ不明なのかを順番に説明する．

1ブロック目の1ラウンド目の算術差分の決め方

1ブロック目の1ラウンド目の算術差分について，Wangらの論文の中では具体的な値は記述されているが，その導出方法についてはほとんど記載されていない．1ブロック目の1ラウンド目の算術差分は2ラウンド目以降の算術差分の決め方とは異なる．これはディスターバンスベクトル$x_{-5}, ..., x_{-1}$の存在と関数fの特徴からローカルコリジョンのルールどおりに算術差分を決定できないからである．

まず，ディスターバンスベクトル$x_{-5}, ..., x_{-1}$について説明する．例えば$x_{-4} = 0x00000002$の場合，ステップ1に$\Delta m_0 = \pm 2^{31}$，ステップ2に$\Delta m_1 = \pm 2^{31}$の差分が発生する．しかし，$x_{-4} = 0x00000002$によるローカルコリジョンを作るためには$\Delta d_0 = \pm 2^{31}$でなければならないが，1ブロック目の初期値はSHA-1の仕様で定められた値を用いるので$\Delta d_0 = 0$になる．この例のように$x_{-5}, ..., x_{-1}$のいずれかが0以外の値であった場合，これらのディスターバンスベクトルから発生する差分はローカルコ

リジョンによって消すことはできない．そのため，このローカルコリジョンによって消すことができない差分は別の 1 ラウンド目の差分で消す必要がある．Wang らの論文には，上記の余分な差分は 1 ラウンド目の関数 f を用いると消すことができると記述されている．また余分な差分を，他のどの差分を用いて消去するかなども記述されている．しかし，消去用の差分の決定法や関数 f の特徴を利用してどの差分をコントロールするかなど一切記述されていない．これらの詳細を明かすことができれば Wang らとは別のディスターバンスベクトルから差分パスを生成することが可能になる．また，この後で記述する 2 ブロック目の 1 ラウンド目の算術差分の話にもつながってくる．

2 ブロック目の 1 ラウンド目の算術差分の決め方

2 ブロック目の 1 ラウンド目の算術差分は Wang らの論文には 1 ブロック目とは異なり例すら記述されていない．この理由は，2 ブロック目の 1 ラウンド目の差分は 1 ブロック目の出力値が決定しないと求めることができないからである．そのため，Wang らの手法でコリジョンを発見するためには 2 ブロック目の 1 ラウンド目の差分値を決定するアルゴリズムが必要になる．しかし，1 ブロック目の 1 ラウンド目の所でも記述したが，1 ラウンド目の差分値の決め方は一切 Wang の論文には記載されていない．そのため，Wang らの手法は，その記述のレベルではコリジョン探索を行うには不十分である．以上のことをさらに詳しく説明すると以下のようになる．

Wang らは 1 ブロック目の出力に関して sufficient condition をつけなくても良いと主張している．Wang らの手法を用いた場合 1 ブロック目の出力差分 Δh_{1block} は以下のようになる．そのため，出力値に sufficient condition をつけない場合，差分の桁上がりをコントロールできない．

$$\Delta h_{1block} = (2^{15}, 2^7\ 2^8, 0, 2^8, 0)$$

1 ブロック目の出力値は 2 ブロック目の初期値として用いられるので，コリジョン探索を実行するごとに 2 ブロック目の初期値の差分によって発生する 2 ブロック目の内部変数の差分をコントロールすることができない．また，2 ブロック目の初期値は 2 ブロック目のステップ 1 からステップ 4 までの計算に用いられるので，2 ブロック目のメッセージ差分から発生する差分をコントロールできない．よって，Wang の手法を用いるとコ

リジョン探索毎に2ブロック目の差分が変化する．すなわち，Wangらの手法を実現するためには，2ラウンド目の差分パスを2^{69}回以下のSHA-1演算で生成するアルゴリズムが必要になる．しかし，Wangらの論文では，2ブロック目の差分パスを生成する方法は示されていない．以上の理由から，Wangらの手法は，その記述のレベルではコリジョン探索を行うには不十分である．

Wangらのディスターバンスベクトルの最適性

SHA-1のディスターバンスベクトルの探索空間は2^{512}と非常に広い空間になっている．そのため，Wangらは効率的なものが含まれていそうな空間に絞り，探索空間を2^{38}に狭めディスターバンスベクトルの探索を行った．しかし，探索空間を絞っているので，探索した範囲以外の空間でさらに効率のいいものが存在する可能性がある．そのため，Wangらの攻撃の限界点を見極めるためにはWangらの探索範囲外のディスターバンスベクトルについても検討を行う必要がある．

ステップ17以降のmessage modification

Wangらはステップ17からステップ22のsufficient conditionをmessage modificationを用いると，このsufficient conditionを満たすように修正することができると主張している．論文ではステップ11からステップ16のメッセージを操作するとできるとだけ記述している．しかし，これ以外のことは記述されておらず，本当にステップ17からステップ22のsufficient conditionを満たすように修正できるmessage modificationかあるかどうかは不明である．2^{69} SHA-0演算でコリジョンを発見するためには，ステップ17からステップ22のsufficient conditionに対するmessageg modificationが必要なので，message modificationの不明な点を明らかにすることが必要である．

図 4.1: 不明な点の概要

第5章 Improved Collision Attack on MD4 with Probability Almost 1

abstract

In EUROCRYPT2005, a collision attack on MD4 was proposed by Wang, Lai, Chen, and Yu. They claimed that collision messages were found with probability 2^{-6} to 2^{-2}, and the complexity was less than 2^8 MD4 hash operations. However, there were some tyops and oversights in their paper. In this paper, first, we reevaluate the exact success probability. Second, we point out the typos and oversights in the paper of Wang et al, and we show how to improve them. Third, we propose a new message modification method for the third round of MD4. From the first result, we reevaluate that the method of Wang et al. can find collision messages with success probability $2^{-5.61}$. From the second result, we can find collision messages with success probability 2^{-2}. Also by combining the second result and the third result, our improved method is able to find collision messages with probability almost 1. This complexity is less than 3 repetitions of MD4 hash operations. Our improved method is about 85 times as fast as the method of Wang et al.

5.1 Introduction

MD4 is a hash function which was proposed by Rivest in CRYPTO'90 [12]. After MD4 was proposed, various hash functions based on MD4 were proposed such as MD5, RIPEMD, SHA-family, and so on. MD4

consists of addition, XOR and bit-rotation, and thus can be calculated fast.

In EUROCRYPT 2005, Wang et al. proposed an efficient attack for MD4 [9]. This attack is a differential attack. While differential attacks before [9], such as [2] or [3] used XOR to calculate differential, the method of Wang et al. uses modular subtraction.

In this method, input differential is decided in advance. Then, the attack will succeed if output differential is 0. The method in [9] makes conditions on chaining variables in order to cancel the input differential. In this paper, we call these conditions on chaining variables introduced to cancel the input differential as "sufficient condition". If an inputted message into MD4 satisfies all the sufficient condition, the output differential becomes 0, and we can generate a collision message. However, the probability that a randomly chosen message satisfies the sufficient condition is very small, and we need more effort to raise this probability.

Message modification methods were proposed in [9] in order to generate a message satisfying the sufficient condition. By applying the combination of these methods, the probability where a message satisfies the sufficient condition is greatly improved. Consequently, Wang et al. claimed that their attack will succeed with probability 2^{-6} to 2^{-2}.

In this paper, we verify the efficiency of [9]. As a result, we found that there were three types of typos and oversights related to success probability.

1. Some messages are appropriately modified if they are modified solely, however, they are modified incorrectly if other specific messages are modified (Discussed in section 4.1.1).

2. Our modification method defined in [9] always fails (Discussed in section 4.1.2).

3. The sufficient condition proposed by Wang et al. does not cover all necessary conditions (Discussed in section 4.2).

Next, we consider these three situations and more precisely evaluate the success probability of their attack. We show that the success probability of the attack by Wang et al. is about $2^{-5.61}$ (Discussed in section 4.3.1).

Second, we correct these typos and oversights, and then, we show this attack succeeds with probability 2^{-2} (Discussed in section 4.3.2). Third, we propose new message modification methods to satisfy the sufficient condition in the third round (Discussed in section 5). By combining these improvement, the success probability can be almost 1. The complexity of our attack can be reduced to 3 repetitions of MD4 hash operations. Our improved method is 85 times as fast as the attack by Wang et al. As far as we know, our attack is the most efficient attack of all the attacks proposed so far.

In EUROCRYPT 2005, Wang et al. proposed the collision attack for MD5, which is the similar attack for MD4 [10]. Since MD5 and MD4 have the similar structure, we expect that our improved method can also be applied to MD5. We will present details of the attack for MD5 later.

5.2 Description of MD4

MD4 is a function which compresses an arbitrary length message into a 128-bit hash value. The hash value of MD4 is calculated by following procedure:

1. Message M is divided into plural words $M_1, M_2, ..., M_n$, where the length of each message block is 512.

2. Let h_i be an output value of i^{th} block. h_i is calculated by using a compression function with h_{i-1} and M_i. This process is repeated until all M_i are calculated. The initial value h_0 is defined as follows;

$$h_0 = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476).$$

3. An output value of the last message block h_n will be the hash value of MD4.

Compression Function. A 512-bit word M_i inputted into the compression function is divided into 32-bit words $m_0, ..., m_{15}$. The compression function consists of 48 steps. Step1 to step16 are called as the first round, similarly, step17 to step32 are the second round and step33

to step48 are the third round. In each step, one of chaining variables a, b, c, d is calculated in order of $a_1, d_1, c_1, b_1, a_2, ...$ Only one variable is newly calculated in each step. The output of the compression function is $h_j = (a_0 + a_{16}, b_0 + b_{16}, c_0 + c_{16}, d_0 + d_{16})$.

We define a function ϕ as follows:

$$\phi(x, y, z, w, m, s, t) = ((x + f(y, z, w) + m + t) \mod 2^{32}) \lll s,$$

where m is a input message in each step, and which m_i is used is defined in advance. s is a number of bit rotation defined in each step. t is a constant defined in each round. In the first round, $t = 0$, in the second round, $t = 0x5a827999$ and in the third round, $t = 0x10325476$. The function f is defined as $f(y, z, w) = F(y, z, w)$ in the first round, $f(y, z, w) = G(y, z, w)$ in the second round and $f(y, z, w) = H(y, z, w)$ in the third round. The function $F(y, z, w)$, $G(y, z, w)$ and $H(y, z, w)$ are defined by,

$$\begin{aligned} F(y, z, w) &= (y \wedge z) \vee (\neg y \wedge w), \\ G(y, z, w) &= (y \wedge z) \vee (y \wedge w) \vee (z \wedge w), \\ H(y, z, w) &= y \oplus z \oplus w. \end{aligned}$$

Chaining variables a_i, b_i, c_i, d_i are calculated as follows:

$$\begin{aligned} a_i &= \phi(a_{i-1}, b_{i-1}, c_{i-1}, d_{i-1}, m, s, t) \\ d_i &= \phi(d_{i-1}, a_i, b_{i-1}, c_{i-1}, m, s, t) \\ c_i &= \phi(c_{i-1}, d_i, a_i, b_{i-1}, m, s, t) \\ b_i &= \phi(b_{i-1}, c_i, d_i, a_i, m, s, t) \end{aligned}$$

5.3 Attack of Wang et al. on MD4

An attack of Wang et al. is a differential attack using modular subtraction to calculate differential. This method makes a collision message by canceling input differential ΔM and making output differential be 0. Usually, the probability that the output differential of two different messages is 0 is very small. In [9], conditions are introduced to chaining variables in order to cancel the input differential.

For example, we consider the case that $\Delta m_1 = 1_{32}$ is given as differential of m_1. Here, 1_i denotes a 32-bit binary number whose i-th bit is 1

and other bits are 0. Because of this differential, the differential $0 \rightarrow 1$ occurs on the 7th bit of d_1, which is a variable calculated in step 2. In this paper, we describe the resulting value of d_1 as $d_1[7]$. In case that the 7th bit would change from 1 to 0, we describe it as $d_1[-7]$.

d_1 is used for calculation of a_2. The calculation for a_2 is as follows.

$$a_2 = (a_1 + F(b_1, c_1, d_1) + m_4) \lll 3$$

Therefore, the differential may be transmitted to a_2. Since this differential should be canceled, we focus on a property of F function. Since the expression of F in this step is $F(b_1, c_1, d_1) = (b_1 \wedge c_1) \vee (\neg b_1 \wedge d_1)$, if $b_{1,7} = 1$, the expression $F(b_1, c_1, d_1) = F(b_1, c_1, d_1[7])$ is true, and thus the change of d_1 can be ignored. Therefore, the differential can be canceled by introducing $b_{1,7} = 1$ in the sufficient condition.

By repeating this process for every differential to introduce all sufficient conditions, the input differential can be canceled. However, the probability that a randomly chosen message satisfies all sufficient conditions is very small. Therefore, it is necessary to try to raise the probability by message modification. In the next section, we explain the message modification.

5.3.1 Technique of Attack of Wang et al.

The attack of Wang et al uses following values as input differential:

$$\Delta M = M^* - M' = (\Delta m_0, \Delta m_1, ..., \Delta m_{15})$$
$$\Delta m_1 = 1_{32}, \Delta m_2 = 1_{32} - 1_{28}, \Delta m_{12} = -1_{17}$$
$$\Delta m_i = 0, 0 \leq i \leq 15, i \neq 1, 2, 12$$

The number of sufficient conditions is 122. Almost all sufficient conditions are satisfied by message modification. However, message modification is adapted only in both the first round and the second round. Therefore, the sufficient condition in the third round is not always satisfied. In this case, it is necessary to rechoose random messages, and start message modification again. Rechosen messages are only m_{14} and m_{15}, and thus operations from step 1 to step 14 are saved. A procedure to generate a collision message pair (M', M^*) is as follows.

1. Select a 512-bit random message M.

2. Modify message M by using message modification to satisfy the sufficient condition.

3. If all sufficient conditions are satisfied, message M' which is a modified message from M is outputted, otherwise m_{14} and m_{15} are rechosen and go back to step 2.

4. Calculate a message $M^* = M' + \Delta M$, and output (M', M^*).

Procedure of Message Modification

In this section, we explain the process of step 2. The purpose of message modification is satisfying the sufficient condition. There are two types of modification techniques. One is single-step modification and the other is multi-step modification. Single-step modification is message modification for the first round. Since this modification does not affect other chaining variables, the single-step modification can be easily done. Multi-step modification is message modification for the second round. Since message modification in the second round affects not only the second round but also the first round, it is necessary to cancel the effect to the first round.

Single-Step Modification

Single-step modification modifies messages in order to satisfy the sufficient condition in the first round. We show an example of the single-step modification. Now, we consider the way to satisfy sufficient conditions in the third step ($c_{1,7} = 1$, $c_{1,8} = 1$, $c_{1,11} = 0$ and $c_{1,26} = d_{1,26}$). The calculation for c_1 is $c_1 = (c_0 + F(d_1, a_1, b_0) + m_2) \lll 11$. In this calculation, m_2 is the only variable which can be changed directly. Therefore, we satisfy sufficient conditions for the third step by modifying m_2 as follows;

$$c_1^{new} \leftarrow c_1 \oplus (c_{1,7} \oplus 1_7) \oplus (c_{1,8} \oplus 1_8) \oplus c_{1,11} \oplus (c_{1,26} \oplus d_{1,26})$$
$$m_2 \leftarrow (c_1^{new} \ggg 11) - c_0 - F(d_1, a_1, b_0).$$

Multi-Step Modification

Multi-step modification modifies messages in the second round in order to satisfy the sufficient condition in the second round. We show an example of the multi-step modification. Now, we consider the way to satisfy $a_{5,19} = c_{4,19}$ which is a sufficient condition in the 17th step. The calculation for a_5 is $a_5 = (a_4 + G(b_4, c_4, d_4) + m_0 + 0x5a827999) \lll 3$. In this calculation, m_0 is the only variable which can be changed directly. Therefore, if $a_{5,19} = c_{4,19}$ is not satisfied, we modify m_0 in order to satisfy this condition in the following way;

$$m_0 \leftarrow m_0 \pm 1_{16}.$$

The condition $a_{5,19} = c_{4,19}$ is satisfied by this modification. However, the change of m_0 causes changes of chaining variables in the first round because m_0 is used not only in the second round but also in the first round. The influence to the first round by modifying m_0 is shown as follows.

$$m_0 \leftarrow m_0 \pm 1_{16} \Rightarrow a_1^{new} = a_1[\pm 19]$$

This influence causes changes of variables which have already satisfied the sufficient condition. Therefore, it is necessary to add following modifications in order to prevent the influence of the change from transmitting to variables in the first round.

$$\begin{aligned}
&m_0 \leftarrow m_0 \pm 1_{16} \Rightarrow a_1^{new} = a_1[\pm 19] \\
&m_1 \leftarrow (d_1 \ggg 7) - d_0 - F(a_1^{new}, b_0, c_0) \\
&m_2 \leftarrow (c_1 \ggg 11) - c_0 - F(d_1, a_1^{new}, b_0) \\
&m_3 \leftarrow (b_1 \ggg 19) - b_0 - F(c_1, d_1, a_1^{new}) \\
&m_4 \leftarrow (a_2 \ggg 3) - a_1^{new} - F(b_1, c_1, d_1)
\end{aligned}$$

By this modification, the influence to the first round is canceled. In this paper, we call this modification as multi-step modification(1).

However, there exists some sufficient conditions in the second round which are failed to be satisfied by the multi-step modification(1). We show an example of this situation. Now, we consider $c_{5,26} = d_{5,26}$ which is a sufficient condition in step 19. If we try to apply the multi-step modification(1), the change of the message affects the first round as follows;

$$m_8 \leftarrow m_8 \pm 1_{18} \Rightarrow a_3^{new} = a_3[\pm 21].$$

表 5.1: Correction method of $c_{5,27}$

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
6	7	$m_5 \leftarrow m_5 + 1_{11}$	$d_2[18], a_2, b_1, c_1$	$d_{2,18} = 0$
7	11		$c_2, d_2[18], a_2, b_1$	$a_{2,18} = b_{1,18}$
8	19		$b_2, c_2, d_2[18], a_2$	$c_{2,18} = 0$
9	3	$m_8 \leftarrow m_8 - 1_{18}$	$a_3, b_2, c_2, d_2[18]$	$b_{2,18} = 0$
10	7	$m_9 \leftarrow m_9 - 1_{18}$	d_3, a_3, b_2, c_2	

Since there exists a sufficient condition about $a_{3,21}$, this modification method breaks the sufficient condition on $a_{3,21}$, and thus we failed to find collision messages. Therefore, we try to apply another message modification method shown in Table 1 in order to keep $a_{3,21}$ unchanged. In Table 1, "extra condition" means conditions of chaining variables introduced to stop the influence in the second round from transmitting to the first round. The extra condition is set to be satisfied when the first round is calculated.

If this modification is applied, the value of the 18th bit of function F in step 9 is changed as follows;

$$F(b_{2,18}, c_{2,18}, d_{2,18}) = 0 \rightarrow 1.$$

The influence to a_3 caused by modifying message m_8 can be canceled by the change of $F(b_{2,18}, c_{2,18}, d_{2,18})$. Therefore, the condition on $c_{5,26}$ is satisfied and the value of $a_{3,20}$ is unchanged by this modification. In this paper, we call this type of modification as multi-step modification(2).

5.4 Exact Evaluation of the Method of Wang et al.

In [9], Wang et al. claimed that success probability of their collision attack on MD4 is 2^{-6} to 2^{-2}. However, this success probability is not enough precise. First, we examined details of the method of Wang et al, and evaluate precise probability.

5.4.1 Oversights in the Method of Wang et al.

Wang et al. proposed various message modification methods. However, as a result of our analysis, we found some of their modifications could not satisfy the sufficient condition. We describe the reason why they could not satisfy the sufficient condition and how to improve the method of Wang et al.

Exclusive Modifications

If some messages are simultaneously modified in order to satisfy specific two sufficient conditions, one modification can conflict the other modification. These cases are a pair of corrections for $d_{5,19} = a_{5,19}$ and $c_{5,26} = d_{5,26}$ and a pair of corrections for $c_{5,32} = d_{5,32}$ and $c_{6,32} = d_{6,32}$. We describe the reason why these corrections are failed, and propose an improved method.

(1) Problems

First, we explain correction of $d_{5,19} = a_{5,19}$ and $c_{5,26} = d_{5,26}$. Wang et al. used the multi-step modification(1) to correct the condition on $d_{5,19}$. In this case, m_4 is modified such as $m_4 \leftarrow m_4 + 1_{14}$. This modification causes change of the value of $a_{2,17}$. Whereas, Wang et al. used the multi-step modification(2) to correct the condition on $c_{5,26}$. In this case, it is necessary to add an extra condition $a_{2,17} = b_{1,17}$. However, if both modification is done, the value of $a_{2,17}$ is changed because of the correction of $d_{5,19}$, and then the extra condition $a_{2,17} = b_{1,17}$ is broken. This extra condition is added in order to cancel the influence of change in the second round. Therefore, if the extra condition is broken, the value of c_2 may change into the different value from desired one.

(2) Success probability of the correction

The above problem occurs when both of $d_{5,19}$ and $c_{5,26}$ are corrected. This probability is 1/4. However, if $d_{6,26}$ is corrected additionally, the

表 5.2: Correction method of $d_{5,19}$

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
2	7	$m_1 \leftarrow m_1 + 1_7$	$d_1[14], a_1, b_0, c_0$	$d_{1,14} = 0$
3	11		$c_1, d_1[14], a_1, b_0$	$a_{1,14} = b_{0,14}$
4	19		$b_1, c_1, d_1[14], a_1$	$c_{1,14} = 0$
5	3	$m_4 \leftarrow m_4 - 1_{14}$	$a_2, b_1, c_1, d_1[14]$	$b_{1,14} = 0$
6	7	$m_5 \leftarrow m_5 - 1_{14}$	d_2, a_2, b_1, c_1	

表 5.3: Correction method of $c_{5,32}$

step	Modify m_i	Chaining value after message modification	Extra Conditions
15	$m_{14} \leftarrow m_{14} + 1_{11}$	$c_4[22], d_4, a_4, b_3$	$c_{4,22} = 0$
16		$b_4, c_4[22], d_4, a_4$	$d_{4,22} = a_{4,22}$
17		$a_5, b_4, c_4[22], d_4$	$b_{4,22} = d_{4,22}$
18		$d_5, a_5, b_4, c_4[22]$	$a_{5,22} = b_{4,22}$
19		$c_5^{new} = c_5 + 1_{31}, d_5, a_5, b_4$	$c_{5,31} = 1$

correction of $d_{5,19}$ and $c_{5,26}$ works appropriately. We explain this mechanism.

When $d_{5,19}$ and $c_{5,26}$ are corrected, and additionally, $d_{6,26}$ is corrected, $c_{5,26}$ is appropriately corrected. The problem is that extra condition $a_{2,17} = b_{1,17}$ is broken, and c_2 will be unexpectedly changed from this influence. However, while $d_{6,26}$ is corrected, m_6 is modified as follows;

$$m_6 \leftarrow (c_2 \ggg 11) - c_1 - f(d_2', a_2, b_1).$$

The purpose of this modification is keeping the value of c_2 unchanged. Therefore, the value of c_2 is kept unchanged even if the extra condition $a_{2,17} = b_{1,17}$ is broken. Therefore, if $d_{5,19}$ and $c_{5,26}$ are corrected and furthermore $d_{6,26}$ is corrected, $c_{5,26}$ is appropriately corrected. Therefore, the success probability of the correction of both $d_{5,19}$ and $c_{5,26}$ is given by

$$1 - (\frac{1}{4} \times \frac{1}{2}) = \frac{7}{8}.$$

Similar situation occurs when both of $c_{5,32}$ and $c_{6,32}$ are corrected. In this situation, the success probability is 1/4.

(3) Improve method

Improved correction methods of $d_{5,19}$ and $c_{5,26}$

Since the method of Wang et al. contains problems explained above, we propose the method modifying $d_{5,19}$ by the multi-step modification(2) in order to avoid the problems. Details of this modification method is shown in Table 2. As a result, the correction of $d_{5,19}$ does not break the extra condition $a_{2,17} = b_{1,17}$, and thus, the success probability of corrections of $d_{5,19}$ and $c_{5,26}$ becomes 1.

Improved correction methods of $c_{5,32}$ and $c_{6,32}$

Since the method of Wang et al. contains problems explained above, we change the correction method of $c_{5,32}$ as shown in Table 3 so that this correction method does not cause problems when both of $c_{5,32}$ and $c_{6,32}$ are corrected.

In the case we change the correction method of $c_{5,32}$, other changes except for Table 3 seem to be possible, for example, modifying message used in 19th step. However, all of other changes which we tried caused influence to other conditions. Therefore, we decided to use carry in the 31st bit in order to change the value of $c_{5,32}$. This modification does not break other conditions. Therefore, the success probability of the corrections of $c_{5,32}$ and $c_{6,32}$ becomes 1.

A modification breaking the sufficient condition

(1) Problems

In [10], $c_{5,29}$ is modified by using the multi-step modification(2). This modification changes the value of $d_{2,20}$ from 0 to 1. However, there exists a sufficient condition $d_{2,20} = a_{2,20}$. Therefore, the modification of $c_{5,29}$ breaks this condition and we fail to get collision messages.

表 5.4: Correction of $c_{5,29}$

step	Modify m_i	Chaining value after message modification	Extra Conditions
15	$m_{14} \leftarrow m_{14} + 1_9$	$c_4[20], d_4, a_4, b_3$	$c_{4,20} = 0$
16		$b_4, c_4[20], d_4, a_4$	$d_{4,20} = a_{4,20}$
17		$a_5, b_4, c_4[20], d_4$	$b_{4,20} = d_{4,20}$
18		$d_5, a_5, b_4, c_4[20]$	$a_{5,20} = b_{4,20}$
19		$c_5^{new} = c_5 + 1_{29}, d_5, a_5, b_4$	

(2) Improved method

c_5 is calculated as follows.

$$c_5 = (c_4 + G(d_5, a_5, b_4) + m_8 + 0x5a827999) \lll 9$$

If we modify m_8 in order to satisfy a sufficient condition $c_{3,29} = 1$, it causes influence to other variables. Therefore, we change the value of $c_{4,20}$ instead. c_4 is calculated by following expression,

$$c_4 = (c_3 + F(d_4, a_4, b_3) + m_{14}) \lll 11.$$

Therefore, we can change $c_{4,20}$ by changing $m_{14,9}$. This correction does not cause influence to other variables, and thus the success probability of the correction of $c_{5,29}$ becomes 1. Table 4 shows details of the correction of $c_{5,29}$

5.4.2 Lack of the Sufficient Condition

We carefully checked the sufficient condition from input differential, and found that conditions $a_{6,30} = 0$ and $b_{4,32} = c_{4,32}$ are lacked in the table 5 written in [9]. In this section, we show how we found these conditions.

Introduction of a sufficient condition "$a_{6,30} = 0$"

At the first,we explain how to find "$a_{6,30} = 0$". Table 5 is a part of table 5 written in [9]. $a_6[30]$ in the Table 5 means that the value of $a_{6,30}$

表 5.5: Introduction of "$a_{6,30} = 0$"

Step	Shift	Δm_i	The i-th output for M'
21	3	2^{31}	$a_6[-29, 30, -32]$

changes from 0 to 1 because of differential. Therefore, we need to set the sufficient condition $a_{6,30} = 0$ in advance. The correction method of $a_{6,30}$ is the same with other correction methods of a_6.

Introduction of a sufficient condition "$b_{4,32} = c_{4,32}$"

表 5.6: Introduction of "$b_{4,32} = c_{4,32}$"

Step	Shift	Δm_i	The i-th output for M'
14	7		$d_4[-27, -29, 30]$
15	11		c_4
16	19		$b_4[19]$
17	3		$a_5[-26, 27, -29, -32]$
18	5		d_5

Table 6 shows that the value of $a_{5,32}$ is changed because of differential, but d_5 is not affected by differential. Therefore, when a_5 is calculated, the differential $a_5[-32]$ has to be canceled. We explain how to cancel this differential. d_5 is calculated by

$$d_5 = ((d_4 + G(a_5, b_4, c_4) + m_4 + 0x5a827999) \mod 2^{32}) \lll 5,$$

where the function G is defined as $G(y, z, w) = (y \wedge z) \vee (y \wedge w) \vee (z \wedge w)$. Since the function G uses a_5, we can cancel differential by arranging other variables used in G. Since G is a majority function, the differential $a_{5,32}$ is canceled by constructing condition $b_{4,32} = c_{4,32}$.

5.4.3 Reevaluation of success probability of the method of Wang et al. and our improved method

Precise probability evaluation of the method of Wang et al.

Considered all the problems explained so far, precise probability of [9] is as follows,

- Success probability considered lack of sufficient conditions $b_{4,32} = c_{4,32}$ and $a_{6,30} = 0$ (section4.2): $\left(\frac{1}{2}\right)^2$
- Success probability considered wrong correction of $c_{5,29}$(section4.2):$\frac{1}{2}$
- Success probability considered problems of corrections of $d_{5,19}$ and $c_{5,26}$ and corrections of $c_{5,32}$ and $c_{6,32}$(section4.1.2):$\frac{7}{8} \times \frac{3}{4}$
- Probability satisfying the sufficient condition in the third round(section4.1.1):$\left(\frac{1}{2}\right)^2$

By combining above analyses, the precise success probability of [9] is given as follows:

$$\begin{aligned} Pr[success] &= \frac{3}{4} \times \frac{7}{8} \times \frac{1}{2} \times \left(\frac{1}{2}\right)^2 \times \left(\frac{1}{2}\right)^2 \\ &= 2^{-5.61} \end{aligned}$$

Furthermore, we experimentally verified this evaluation. In our experiment, we generated 5,000 collision messages by the method of Wang et al, and the average number where we needed to rechoose random messages to find a collision was 48.76 times($\fallingdotseq 2^{5.60}$ times). Therefore, the theoretical result is consistent with the experimental result.

Our Improved Method

We improved oversights and typos of [9] as follows,

- Add two sufficient conditions $b_{4,32} = c_{4,32}$ and $a_{6,30} = 0$
- Change the modification method of $d_{5,19}$ so that both of $d_{5,19} = a_{5,19}$ and $c_{5,26} = d_{5,26}$ can be corrected.

表 5.7: Shortcut Modification

step	Modify m_j	Chaining value after message modification	Extra Conditions in 1^{st} round
12	$m_{11} \leftarrow m_{11} \pm 1_i$	$b_3[\pm(i+19)], c_3, d_3, a_3$	
13		$a_4, b_3[\pm(i+19)], c_3, d_3$	$c_{3,i+19} = d_{3,i+19}$
14		$d_4, a_4, b_3[\pm(i+19)], c_3$	$a_{4,i+19} = 0$
15		$c_4, d_4, a_4, b_3[\pm(i+19)]$	$d_{4,19} = 0$
16	$m_{15} \leftarrow (b_4 \ggg 19) - b_3[\pm(i+19)] - F(c_4, d_4, a_4)$	b_4, c_4, d_4, a_4	

- Change the modification method of $c_{5,32}$ so that both of $c_{5,32} = d_{5,32}$ and $c_{6,32} = d_{6,32} + 1$ can be corrected.

By our improvement, all sufficient conditions in both the first round and the second round can be corrected with probability 1. Therefore, the success probability becomes exactly 2^{-2}.

In [9], details of the multi-step modification is not written. In this paper, we attach the list of all multi-step modification methods in appendix.

5.5 Shortcut Modification

5.5.1 Our idea

In the case of multi-step modification, only the first round is affected by message modifications. Therefore, to make the multi-step modification succeed, only the first round has to be considered. If we try to apply same approach to modify messages in the third round, we have to consider the influence of the modification about both the first round and the second round. This is very difficult. Actually, the message modification for the third round is not mentioned in [9]. To overcome this difficulty, we randomly modify messages in the second round many times in order to satisfy the sufficient condition in the third round. In the method of Wang et al, if collision message cannot be generated, algorithm returns to the calculation for the first round, rechooses m_{14} and m_{15} and apply the multi-step modification in the second round. In the proposed method, we change few bits of a message in the second round instead of rechoosing

random messages. Therefore, the complexity of each repetition is reduced by the complexity for the multi-step modification.

5.5.2 Consideration

Here, we consider to modify m_{11}. To begin with, we experimentally found probability that the sufficient condition in the third round is satisfied by modifying m_{11} with 1 bit. Hence, we can assume that conditions are satisfied with probability about 1/4 for all single bit change. When we modify m_{11} in the second round, we have to cancel the influence to the first round caused by modification of m_{11}. We apply message modification shown in Table 7 to cancel the influence from m_{11}. The number of i, where the extra conditions does not break the sufficient condition is 19. If we simultaneously change several bits of m_{11}, we can similarly assume that conditions are satisfied with probability 1/4. Therefore, the number of repetition by modifying m_{11} is equal to the number of all combination of 19 bits, that is, 2^{19}. If we modify m_{11} many times, probability to satisfy the sufficient conditions in the third round becomes very high. This probability is given by

$$1 - \left(1 - \frac{1}{4}\right)^{2^{19}} \fallingdotseq 1.$$

Therefore, the probability that the sufficient condition in the third round becomes almost 1. Since this probability 1/4 for each m_{11}, the average number that we change m_{11} to satisfy conditions in the third round is three. (First time, we calculate m_{11} in the standard way. Then, if we repeat the process three times, four of different m_{11} would be tried.)

Remark 1

The reason we chose m_{11} is that since m_{11} is the second last message in the second round, the number of steps of each repetition is small compared to other messages. The reason we did not choose m_{15} is that we could not cancel the influence to the first round caused by m_{15}.

5.6 Our Improvement

In section 4 and 5, we proposed the message modification which satisfies the sufficient condition with probability almost 1. The followings are our improvement,

- We add the sufficient condition $b_{4,32} = c_{4,32}$ and $a_{6,30} = 0$
- We change the message modification method of $d_{5,19}$ in order to enable both $d_{5,19} = a_{5,19}$ and $c_{5,26} = d_{5,26}$ to be modified.
- We change the message modification method of $c_{5,32}$ in order to enable both $c_{5,32} = d_{5,32}$ and $c_{6,32} = d_{6,32} + 1$ to be modified.
- We propose the shortcut modification which raise the probability satisfying the sufficient condition in the third round.

In our improved method, collision message can be generated with probability almost 1. The complexity of our method is given by,

- 1 MD4 operation to get hash value of the message in standard calculation
- Few steps for the single-step modification
- 26 steps for the multi-step modification and 13 steps for recalculation of chaining variables after the multi-step modification is applied
- 24 steps for modification of m_{11} to satisfy the sufficient condition in the third round (the average number of repetition of changing m_{11} is 3 (section5.2) and each repetition needs 8 steps)
- Few steps to cancel the influence of m_{11}

Therefore, total complexity is 1 MD4 + 26 steps + 13 steps + 24 steps + few steps = 1 MD4 + (63 + few) steps. Since one MD4 operation consists of 48 steps, This complexity is less than three repetitions of MD4 operations. Consequently, collision messages can be generated with less than three repetitions of MD4 operations.

The method of Wang et al. finds collision with complexity less than 2^8 MD4 operations. Whereas, the complexity of our method is 3 MD4

表 5.8: An example of generated collision pair

M	0x24ce9d37de4dfca0a3b88fc39c9f9e5c92ee86ada2c9e8b088f3a020c5368a69 0e503cc80c2368f978ff57bf21a1762ad018afb8daa431e9308bf382806a18a1
M'	0x368b9d377e2dfc60b5b88fcb0c8fbe5601a6662d9ecc3929aa35aabf887f929f 2740a2c8c8c12039bbb401bdc1983331e45e1f61c150d565ee27d04af1dfec4c
M^*	0x368b9d37fe2dfc6025b88fcb0c8fbe5601a6662d9ecc3929aa35aabf887f929f 2740a2c8c8c12039bbb401bdc1983331e45d1f61c150d565ee27d04af1dfec4c
$H(M')$	0x26a280327c3068532de33b679d022e59
$H(M^*)$	0x26a280327c3068532de33b679d022e59

operations. Therefore, our method is $2^8/3 \fallingdotseq 85$ times as fast as the method of Wang et al.

Remark 2

It is possible to reduce the complexity of the first round by randomly generating chaining variables in the first round instead of messages. First, set each chaining variables in the first round to satisfy the sufficient condition. Second, randomly generate other bits. Finally, calculate the message from generated chaining variables. This process reduces the complexity of the first round.

5.7 Conclusion

Wang et al. claimed that collisions of MD4 can be generated with complexity less than 2^8 MD4 hash operations. However, this evaluation was not enough precise. In this paper, we reevaluated the success probability of the method of Wang et al. Then, we correct typos and oversights of [9]. As a result, the attack succeeded with probability 2^{-2}. At the last, we proposed the method to satisfy the sufficient condition in the third round. As a result, the complexity to generate collision messages was reduced to 3 repetitions of MD4 operations.

We show a message, where the method of Wang et al. cannot generate a collision pair. In Table 8, M is the message before modified, M' is the message after modified and M^* is calculated by $M^* = M' + \Delta M$. $H(M')$ is the hash value of M' and $H(M^*)$ is the hash value of M^*.

Appendix(*List of Multi-Step Modification Methods*)

表 5.9: Modification of "$a_{5,i}$" $(i = 19, 26, 27, 29, 32)$

step	shift	Modify m_i	Chaining value after after message modification
1	3	$m_0 \leftarrow m_0 \pm 1_{i-3}$	$a_1^{new} = a_1[\pm i], b_0, c_0, d_0$
2	7	$m_1 \leftarrow (d_1 \ggg 7) - d_0 - F(a_1^{new}, b_0, c_0)$	d_1, a_1^{new}, b_0, c_0
3	11	$m_2 \leftarrow (c_1 \ggg 11) - c_0 - F(d_1, a_1^{new}, b_0)$	c_1, d_1, a_1^{new}, b_0
4	19	$m_3 \leftarrow (b_1 \ggg 19) - b_0 - F(c_1, d_1, a_1^{new})$	b_1, c_1, d_1, a_1^{new}
5	3	$m_4 \leftarrow (a_2 \ggg 3) - a_1^{new} - F(b_1, c_1, d_1)$	a_2, b_1, c_1, d_1

表 5.10: Modification of "$d_{5,19}$"

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
2	7	$m_1 \leftarrow m_1 + 1_7$	$d_1[14], a_1, b_0, c_0$	$d_{1,14} = 0$
3	11		$c_1, d_1[14], a_1, b_0$	$a_{1,14} = b_{0,14}$
4	19		$b_1, c_1, d_1[14], a_1$	$c_{1,14} = 0$
5	3	$m_4 \leftarrow m_4 - 1_{14}$	$a_2, b_1, c_1, d_1[14]$	$b_{1,14} = 0$
6	7	$m_5 \leftarrow m_5 - 1_{14}$	d_2, a_2, b_1, c_1	

表 5.11: Modification of "$d_{5,i}$" $i = 16, 17, 29, 32$

step	shift	Modify m_i	Chaining value after message modification
5	3	$m_4 \leftarrow m_4 \pm 1_{i-5}$	$a_2^{new} = a_2[\pm(i-2)], b_1, c_1, d_1$
6	7	$m_5 \leftarrow (d_2 \ggg 7) - d_1 - F(a_2^{new}, b_1, c_1)$	d_2, a_2^{new}, b_1, c_1
7	11	$m_6 \leftarrow (c_2 \ggg 11) - c_1 - F(d_2, a_2^{new}, b_1)$	c_2, d_2, a_2^{new}, b_1
8	19	$m_7 \leftarrow (b_2 \ggg 19) - b_1 - F(c_2, d_2, a_2^{new})$	b_2, c_2, d_2, a_2^{new}
9	3	$m_8 \leftarrow (a_3 \ggg 3) - a_2^{new} - F(b_2, c_2, d_2)$	a_3, b_2, c_2, d_2

表 5.12: Modification of "$c_{5,i}$" $i = 26, 27$

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
6	7	$m_5 \leftarrow m_5 + 1_{i-16}$	$d_2[i-9], a_2, b_1, c_1$	$d_{2,i-9} = 0$
7	11		$c_2, d_2[i-9], a_2, b_1$	$a_{2,i-9} = b_{1,i-9}$
8	19		$b_2, c_2, d_2[i-9], a_2$	$c_{2,i-9} = 0$
9	3	$m_8 \leftarrow m_8 - 1_{i-9}$	$a_3, b_2, c_2, d_2[i-9]$	$b_{2,i-9} = 0$
10	7	$m_9 \leftarrow m_9 - 1_{i-9}$	d_3, a_3, b_2, c_2	

表 5.13: Modification of "$c_{5,29}$"

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
15	11	$m_{14} \leftarrow m_{14} + 1_9$	$c_4[20], d_4, a_4, b_3$	$c_{4,20} = 0$
16	19		$b_4, c_4[20], d_4, a_4$	$d_{4,20} = a_{4,20}$
17	3		$a_5, b_4, c_4[20], d_4$	$b_{4,20} = d_{4,20}$
18	5		$d_5, a_5, b_4, c_4[20]$	$a_{5,20} = b_{4,20}$
19	9		$c_5^{new} = c_5 + 1_{29}, d_5, a_5, b_4$	

表 5.14: Modification of "$c_{5,30}$"

step	shift	Modify m_i	Chaining value after after message modification
9	3	$m_8 \leftarrow m_8 \pm 1_{21}$	$a_3^{new} = a_3[\pm 24], b_2, c_2, d_2$
10	7	$m_9 \leftarrow (d_3 \ggg 7) - d_2 - F(a_3^{new}, b_2, c_2)$	d_3, a_3^{new}, b_2, c_2
11	11	$m_{10} \leftarrow (c_3 \ggg 11) - c_2 - F(d_3, a_3^{new}, b_2)$	c_3, d_3, a_3^{new}, b_2
12	19	$m_{11} \leftarrow (b_3 \ggg 19) - b_2 - F(c_3, d_3, a_3^{new})$	b_3, c_3, d_3, a_3^{new}
13	3	$m_{12} \leftarrow (a_4 \ggg 3) - a_3^{new} - F(b_3, c_3, d_3)$	a_3, b_3, c_3, d_3

表 5.15: Modification of "$c_{5,31}$"

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
15	11	$m_{14} \leftarrow m_{14} + 1_{11}$	$c_4[22], d_4, a_4, b_3$	$c_{4,22} = 0$
16	19		$b_4, c_4[22], d_4, a_4$	$d_{4,22} = a_{4,22}$
17	3		$a_5, b_4, c_4[22], d_4$	$b_{4,22} = d_{4,22}$
18	5		$d_5, a_5, b_4, c_4[22]$	$a_{5,22} = b_{4,22}$
19	9		$c_5^{new} = c_5 + 1_{31}, d_5, a_5, b_4$	$c_{5,31} = 1$

表 5.16: Modification of "$b_{5,i}$" $i = 29, 32$

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
11	11	$m_{10} \leftarrow m_{10} + 1_{i-24}$	$c_3[-(i-13)], d_3, a_3, b_2$	$c_{3,i-13} = 0$
12	19		$b_3, c_3[-(i-13)], d_3, a_3$	$d_{3,i-13} = a_{3,i-13}$
13	3	$m_{12} \leftarrow m_{12} - 1_{i-13}$	$a_4, b_3, c_3[-(i-13)], d_3$	$b_{3,i-13} = 1$
14	7		$d_4, a_4, b_3, c_3[-(i-13)]$	$a_{4,i-13} = 1$
15	11	$m_{14} \leftarrow m_{14} - 1_{i-13}$	c_4, d_4, a_4, b_3	

表 5.17: Modification of "$b_{5,30}$"

step	shift	Modify m_i	Chaining value after message modification	Extra Conditions
12	19	$m_{11} \leftarrow m_{11} + 1_{30}$	$b_3[17], c_3, d_3, a_3$	$b_{3,17} = 0$
13	3	$m_{12} \leftarrow m_{12} - 1_{17}$	$a_4, b_3[17], c_3, d_3$	
14	7		$d_4, a_4, b_3[17], c_3$	$a_{4,17} = 0$
15	11		$c_4, d_4, a_4, b_3[17]$	$d_{4,17} = 1$
16	19	$m_{15} \leftarrow m_{15} - 1_{17}$	b_4, c_4, d_4, a_4	

表 5.18: Modification of "$a_{6,i}$" $i = 29, 30, 32$

step	shift	Modify m_i	Chaining value after after message modification	Extra Conditions
2	7	$m_1 \leftarrow m_1 \pm 1_{i-3}$	$d_1^{new} = d_1[\pm(i+4)], a_1, b_0, c_0$	
3	11	$m_2 \leftarrow (c_1 \ggg 11) - c_0 - F(d_1^{new}, a_1, b_0)$	c_1, d_1^{new}, a_1, b_0	
4	19	$m_3 \leftarrow (b_1 \ggg 19) - b_0 - F(c_1, d_1^{new}, a_1)$	b_1, c_1, d_1^{new}, a_1	
5	3		a_2, b_1, c_1, d_1^{new}	$b_{1,i+4} = 1$
6	7	$m_5 \leftarrow (d_2 \ggg 7) - d_1^{new} - F(a_2, b_1, c_1)$	d_2, a_2, b_1, c_1	

表 5.19: Modification of "$d_{6,29}$"

step	shift	Modify m_i	Chaining value after after message modification	Extra Conditions
6	7	$m_5 \leftarrow m_5 \pm 1_{24}$	$d_2^{new} = d_2[\pm 31], a_2, b_1, c_1$	
7	11	$m_6 \leftarrow (c_2 \ggg 11) - c_1 - F(d_2^{new}, a_2, b_1)$	c_2, d_2^{new}, a_2, b_1	
8	19	$m_7 \leftarrow (b_2 \ggg 19) - b_1 - F(c_2, d_2^{new}, a_2)$	b_2, c_2, d_2^{new}, a_2	
9	3		a_3, b_2, c_2, d_2^{new}	$b_{2,31} = 1$
10	7	$m_9 \leftarrow (d_3 \ggg 7) - d_2^{new} - F(a_3, b_2, c_2)$	d_3, a_3, b_2, c_2	

表 5.20: Modification of "$c_{6,i}$" $i = 29, 30$

step	shift	Modify m_i	Chaining value after after message modification	Extra Conditions
10	7	$m_9 \leftarrow m_9 \pm 1_{i-9}$	$d_3^{new} = d_3[\pm(i-2)], a_3, b_2, c_2$	
11	11	$m_{10} \leftarrow (c_3 \ggg 11) - c_2 - F(d_3^{new}, a_3, b_2)$	c_3, d_3^{new}, a_3, b_2	
12	19	$m_{11} \leftarrow (b_3 \ggg 19) - b_2 - F(c_3, d_3^{new}, a_3)$	b_3, c_3, d_3^{new}, a_3	
13	3		a_4, b_3, c_3, d_3^{new}	$b_{3,i-2} = 1$
14	7	$m_{13} \leftarrow (d_4 \ggg 7) - d_3^{new} - F(a_4, b_3, c_3)$	d_4, a_4, b_3, c_3	

表 5.21: Modification of "$c_{6,32}$"

step	shift	Modify m_i	Chaining value after after message modification	Extra Conditions
7	11	$m_6 \leftarrow m_6 \pm 1_{12}$	$c_2[23], d_2, a_2, b_1$	$c_{2,23} = 0$
8	19		$b_2, c_2[23], d_2, a_2$	$d_{2,23} = a_{2,23}$
9	3		$a_3, b_2, c_2[23], d_2$	$b_{2,23} = 0$
10	7	$m_9 \leftarrow m_9 - 1_{23}$	$d_3, a_3, b_2, c_2[23]$	
11	11	$m_{10} \leftarrow m_{10} - 1_{23}$	c_3, d_3, a_3, b_2	

第6章 Improved Collision Attack on MD5

abstract

In EUROCRYPT2005, a collision attack on MD5 was proposed by Wang et al. In this attack, conditions which are sufficient to generate collisions (called "sufficient condition") are introduced. This attack raises the success probability by modifing messages to satisfy these conditions. In this attack, 37 conditions cannot be satisfied even messages are modified. Therefore, the complexity is 2^{37}. After that, Klima improved this result. Since 33 conditions cannot be satisfied in his method, the complexity is 2^{33}.

In this paper, we propose new message modification techniques which are more efficient than attacks proposed so far. In this method, 29 conditions cannot be satisfied. However, this method is probabilistic, and the probability that this method work correctly is roughly 1/2. Therefore, the complexity of this attack is 2^{30}. Furthermore, we propose a more efficient collision search algorithm than that of Wang et al. By using this algorithm, the total complexity is reduced into roughly 5/8.

6.1 Introduction

MD5 is one of the hash fucntions which compress an arbitrary length message into a defined length random message. (In case of MD5, the output is 128-bit.) Since hash functions are composed of addition, XOR, bit-rotation and other light operations, they can be calculated quickly. One of the important properties of hash functions is called collision resistance. Let x be a message and $h(x)$ be a hash value of x. Collision

resistance means that it is difficult to find a pair of message (x, y), where $h(x) = h(y)$.

In 1992, Rivest proposed a hash function named MD4 [12]. After that, in 1992, MD5 [13] which is an improved version of MD4 was proposed by Rivest. Then various hash functions based on MD4 such as RIPEMD, SHA-0 and SHA-1 were proposed.

In 1996, Dobbertin proposed a collision attack on MD5 [11]. However, since this attack used modified initial value, this attack was not real attack for MD5. In 2005, Wang et al. proposed an efficient collision attack on MD5 [10]. This attack is a kind of differential attack. Although almost all differential attacks use XOR operation to calculate differential value, Wang et al. uses modular subtraction instead of XOR. In the method of Wang et al, the input differenatial is decided in advance, and the goal of the attack is finding messages which cancel the input differential and make a differential of the hashed values of input messages 0. This attack introduces conditions on chaining variablesto make the output differential 0. These conditions are called "sufficient condition." If an input message satisfies all conditions, collision messages can deterministically be generated. According to [10], the number of conditions is 290. Therefore, the probability that a randomly message satisfies all conditions is 2^{-290}, and this is very small. However, this probability can be improved by modifying a message to satisfy conditions. In this attack, message modification techniques which satisfy except for 37 conditions are proposed. Therefore, the complexity of this attack is 2^{37}.

In 2005, Klima improved the attack of Wang et al. [15]. Message modification techniques discribed in [15] can satisfy except for 33 conditions. Therefore, the complexity of this attack is 2^{33}. In CRYPTO2005, Wang et al. represented an attack to SHA-1 [14]. In this paper, they claimed that they found a technique to generate collisions of MD5 with complexity 2^{32} though details are not mentioned.

In this paper, we propose new message modification techniques which can satisfy except for 29 conditions with probability 1/2. Therefore, the complexity of our attack is 2^{30}. We also propose an efficient collision search algorithm which can reduce the complexity into roughly 5/8 compared to an algorithm of Wang et al.

6.2 Description of MD5

MD5 is a compression function which calculates a 128-bit random value from an arbitrary length message. When a message M is inputted, the hash value of M is calculated by the following way:

1. A message M is devided into 512-bit messages.

$$M = (M_0, M_1, ..., M_n),\ |M_i| = 512$$

2. Let h_i be the output of i-th operation. h_i is calculated by the MD5 compression function with M_{i-1} and h_{i-1}. Repeat this process from M_0 to M_n.

   The initial value h_0 is defined as follows,

   $h_0 = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)$.

3. The output of the operation for the last message is the hash value of M.

Compression Function

A message M_j is compressed by the compression function. This operation is called the $(j+1)$-th block operation. Each block consists of 64 steps. Step 1 to step 16 are called the first round, step 17 to step 32 are called the second round, step 33 to step 48 are called the third round and step 49 to step 64 are called the forth round. In each step, one of the chaining variables a, b, c, d is calculated. The order of the calculated chaining variables is $a_1, d_1, c_1, b_1, a_2, \cdots$. The output of each block h_i is $h_i = (a_0 + a_{16}, b_0 + b_{16}, c_0 + c_{16}, d_0 + d_{16})$.

Chaining variables are calculated in the following way. First, M_j is devieded into 32-bit messages $m_0, \ldots, m_{15}$. Then, a function ψ is defined as follows,

$$\psi(x, y, z, w, m, s, t) = y + ((x + \phi(y, z, w) + m + t) \mod 2^{32}) \lll s,$$

where m denotes a message inputted in that step. Which m is inputted is defined in advance. s denotes a number for left cyclic shift defined in each step. t denotes a constant number defined in each step. ϕ denotes a non-linear function defined in each round. Details of the function ϕ are listed below.

$$
\begin{aligned}
1R: &\quad \phi(y,z,w) = (y \wedge z) \vee (\neg y \wedge w),\\
2R: &\quad \phi(y,z,w) = (y \wedge w) \vee (z \wedge \neg w),\\
3R: &\quad \phi(y,z,w) = y \oplus z \oplus w,\\
4R: &\quad \phi(y,z,w) = z \oplus (y \vee \neg w).
\end{aligned}
$$

Chaining variables a_i, b_i, c_i, d_i are calculated by the following expression.

$$
\begin{aligned}
a_i &= \psi(a_{i-1}, b_{i-1}, c_{i-1}, d_{i-1}, m, s, t),\\
d_i &= \psi(d_{i-1}, a_i, b_{i-1}, c_{i-1}, m, s, t),\\
c_i &= \psi(c_{i-1}, d_i, a_i, b_{i-1}, m, s, t),\\
b_i &= \psi(b_{i-1}, c_i, d_i, a_i, m, s, t).
\end{aligned}
$$

6.3 Description of the Attack by Wang et al. [10]

6.3.1 Notation

We define notations which are used in this paper. $M = (m_0, ..., m_{15})$ and $M' = (m'_0, ..., m'_{15})$ are messeages where each m_i is 32-bit integer. $\Delta M = (\Delta m_0, ..., \Delta m_{15})$ is the differential of M and M', where $\Delta m_i = m'_i - m_i$. $a_i, b_i, c_i, d_i (1 \leq i \leq 16)$ represent chaining variables, and $a_{i,j}, b_{i,j}, c_{i,j}, d_{i,j} (1 \leq j \leq 32)$ represent the j-th bit of respective chaining variables. ϕ_i is the output of the nonlinear function ϕ in the i-th step, and $\phi_{i,j}$ denotes the j-th bit of ϕ_i. $x_i[j]$ and $x_i[-j]$ represent differentials of the j-th bit in chaining variable x_i. $x_i[j]$ represents that differential of the j-th bit in x_i is 1, that is, $x'_{i,j} - x_{i,j} = 1$. Whereas, $x_i[-j]$ represents that differential of the j-th bit in x_i is -1, that is, $x'_{i,j} - x_{i,j} = -1$.

6.3.2 Collision Differentials

The attack of Wang et al. is a differential attack. It generated collision messages by canceling the input differential, and by making output differential 0. It generated collision messages which are composed of two 1024-bit messages (M_0, M_1) and (M_0', M_1'). In this attack, a collision differential is given in advance. The differential is as follows,

$$\Delta H_0 = 0 \overset{(M_0, M_0')}{\longrightarrow} \Delta H_1 \overset{(M_1, M_1')}{\longrightarrow} \Delta H = 0,$$

where,

$$\begin{aligned}
\Delta M_0 &= M_0' - M_0 \\
&= (0,0,0,0,2^{31},0,0,0,0,0,0,2^{15},0,0,2^{31},0), \\
\Delta M_1 &= M_1' - M_1 \\
&= (0,0,0,0,2^{31},0,0,0,0,0,0,-2^{15},0,0,2^{31},0), \\
\Delta H_1 &= (2^{31}, 2^{31}+2^{25}, 2^{31}+2^{25}, 2^{31}+2^{25}).
\end{aligned}$$

6.3.3 Sufficient Condition

Sufficient condition is a set of conditions of chaining variables for generating collisions. If all conditions for compressing M are satisfied, M and $M + \Delta M$ becomes a collision pair with probability 1. According to [10], there are 290 conditions for M_0, therefore, the probability that a randomly chosen message satisfies all conditions is very small. However, it is possible to drastically raise the probability by modifying a message (Described in 3.5). Conditions in the first round and in the early steps of the second round are able to be corrected by modifying a message. However, 37 conditions are not able to be corrected. Therefore, if a random message does not satisfy all of these conditions, it is necessary to choose another random message and modify it again.

6.3.4 Collision Search Algorithm

An algorithm for searching collisions is as follows.

1. Generate a 512-bit random message M_0 for the first block.

2. Modify M_0 in order to satisfy the sufficient condition for the first block as much as possible.

3. If all conditions are satisfied, calculate the output of the first block with the modified message and the MD5 initial value. If all conditions are not satisfied, randomly rechoose m_{14} and m_{15}, and go back to (2).

4. Generate a 512-bit random message M_1 for the second block.

5. By the similar way to the first block, calculate the output of the second block. However, in this time, use the result of the first block instead of using the MD5 initial value.

6. Calculate $M_0' = M_0 + \Delta M_0$, $M_1' = M_1 + \Delta M_1$

7. (M_0, M_1) and (M_0', M_1') are the collision messages.

6.3.5 Message Modification

The purpose of message modification is deterministically (or with very high probability) satisfying the sufficient condition. There are two kinds of message modification:"single-message modification" and "multi-message modification". Single-message modification is a message modification for the first round, and multi-message modification is for the second round. Single-message modification modifies only one message, whereas, multi-message modification affects some messages. Therefore, to do multi-message modification, we need additional work in order to cancel the influence to other messages.

Single-Message Modification

Single-message modification is a message modification for satisfying the sufficient condition in the first round.

For example, we consider the situation where we modify a message in order to satisfy conditions in the third step of the first round. According to [10], there are three conditions on c_1: "$c_{1,8} = 0$","$c_{1,12} = 0$" and

"$c_{1,20} = 0$". On the other hand, the expression for calculating c_1 is as follows,

$$c_1 = (d_1 + (c_0 + \phi(d_1, a_1, b_0) + m_2 + t) \lll 17) \bmod 2^{31}.$$

In order to guarantee that these conditions on c_1 are satisfied, we modify m_2 by the following way.

1. Generate a 32-bit random message m_2^{old}, and calculate the value of c_1^{old}.

2. Change the value of c_1^{old} into desirable c_1^{new} by the expression below.

$$c_1^{new} \leftarrow c_1^{old} - c_{1,7}^{old} \cdot 2^6 - c_{1,12}^{old} \cdot 2^{11} - c_{1,20}^{old} \cdot 2^{19}$$

3. Rewrite the m_2 in order to guarantee that c_1^{new} is always obtained.

$$m_2^{new} \leftarrow ((c_1^{new} - d_1) \ggg 17) - c_0 - \phi(d_1, a_1, b_0) - t$$

The similar procedure is applied to all steps in the first round, and all conditions in the first round are satisfied with high probability.

Multi-Message Modification

Multi-message modification is a message modification for satisfying the sufficient condition in the second round. Some of the conditions in early steps of the second round can be corrected by this modification.

For example, we consider the situation where the condition on $a_{5,32}$ needs to be corrected. According to [10], the condition on $a_{5,32}$ is $a_{5,32} = 0$. Therefore, if the value of $a_{5,32}$ is happen to be 0, $a_{5,32}$ does not have to be corrected. However, if $a_{5,32} = 1$, $a_{5,32}$ has to be corrected by the following way:

First, a_5 is calculated as follows,

$$a_5 \leftarrow b_4 + (a_4 + \phi(b_4, c_4, d_4) + m_1 + t) \lll 5$$

1. Since m_1 is used to calculate a_5, and the bit rotation number in this step is 5, we modify the 27-th bit of m_1 in order to change $a_{5,32}$.

2. Since m_1 is used not only in the second round but in the first round, the change on m_1 affects other steps in the first round. Therefore, we need additional change in the first round.

Details of the correcting $a_{5,32}$ are listed in Table 1. Wang et al. claimed that except for 37 conditions can be corrected by single-message modifications and multi-message modifications.

表 6.1: The message modification for correcting "$a_{5,32}$"

step	m	shift	Modify m_i	Chaining variables
2	m_1	12	$m_1 \leftarrow m_1 + 2^{26}$	d_1^{new}, a_1, b_0, c_0
3	m_2	17	$m_2 \leftarrow ((c_1 - d_1^{new}) \ggg 17) - c_0 - \phi(d_1^{new}, a_1, b_0) - t$	c_1, d_1^{new}, a_1, b_0
4	m_3	22	$m_3 \leftarrow ((b_1 - c_1) \ggg 22) - b_0 - \phi(c_1, d_1^{new}, a_1) - t$	b_1, c_1, d_1^{new}, a_1
5	m_4	7	$m_4 \leftarrow ((a_2 - b_1) \ggg 7) - a_1 - \phi(b_1, c_1, d_1^{new}) - t$	a_2, b_1, c_1, d_1^{new}
6	m_5	12	$m_5 \leftarrow ((d_2 - a_2) \ggg 12) - d_1^{new} - \phi(a_2, b_1, c_1) - t$	d_2, a_2, b_1, c_1

6.4 New Multi-Message Modification

Wang et al. claimed that except for 37 conditions can be corrected. This means that they succeeded in correcting 6 conditions in the second round. In our research, we found that 14 conditions in the second round could be corrected. In this section, we explain which conditions can be corrected and how to correct those conditions.

6.4.1 Extra Condition

To correct many conditions in the second round, it is necessary to make "extra condition". Therefore, we first explain the extra condition. The terminology of the "extra condition" was introduced by Wang et al. [9]. The purpose of setting extra condition is to guarantee that conditions in the second round can be corrected. Extra conditions are not conditions for generating collisions, therefore, messages do not have to satisfy extra

conditions after they are modified in order to correct unsatisfied sufficient conditions.

Extra conditions in the first round are set together with the sufficient condition in the first round by the single-message modification. Extra conditions in the second round are set by the multi-message modification. A collision search algorithm for the first block including the extra condition is as follows,

1. Generate a random message.
2. Modify the message to satisfy the sufficient condition and the extra condition in the first round by single-message modification.
3. If the sufficient condition and the extra condition in the second round are not satisfied, correct them by multi-message modification.
4. If modified message does not satisfy all of the sufficient condition, choose m_{14} and m_{15} again, and go back to (2), otherwise, output the calculated value.

A collision search algorithm for the second block is almost same with the first block.

6.4.2 Details of the Multi-Message Modification

In our research, we have found that 14 conditions in the second round are able to be corrected. Correctable conditions are $a_{5,4}$, $a_{5,16}$, $a_{5,18}$, $a_{5,32}$, $d_{5,18}$, $d_{5,30}$, $d_{5,32}$, $c_{5,18}$, $c_{5,32}$, $b_{5,32}$, $a_{6,18}$, $a_{6,32}$, $d_{6,32}$ and $c_{6,32}$.

Corrections for $a_{5,i}(i = 4, 16, 18, 32)$
a_5 is calculated by the following expression.

$$a_5 = b_4 + (a_4 + \phi(b_4, c_4, d_4) + m_1 + t) \lll 5$$

The i-th bit of a_5 can be corrected by the message modification shown in Table 2.

In Table 2, '$\pm$' is chosen depending on the value of a chaining variable in the first round. Since m_1 is used to calculate d_1 in step 2, the value

表 6.2: The message modification for correcting "$a_{5,i}$"

step	shift	Modify m_i	Chaining Variables
2	12	$m_1 \leftarrow m_1 \pm 2^{i-6}$	$d_1[\pm(i+7)], a_1, b_0, c_0$
3	17	$m_2 \leftarrow ((c_1 - d_1^{new}) \ggg 17) - c_0 - \phi(d_1^{new}, a_1, b_0) - t$	$c_1, d_1[\pm(i+7)], a_1, b_0$
4	22	$m_3 \leftarrow ((b_1 - c_1) \ggg 22) - b_0 - \phi(c_1, d_1^{new}, a_1) - t$	$b_1, c_1, d_1[\pm(i+7)], a_1$
5	7	$m_4 \leftarrow ((a_2 - b_1) \ggg 7) - a_1 - \phi(b_1, c_1, d_1^{new}) - t$	$a_2, b_1, c_1, d_1[\pm(i+7)]$
6	12	$m_5 \leftarrow ((d_2 - a_2) \ggg 12) - d_1^{new} - \phi(a_2, b_1, c_1) - t$	d_2, a_2, b_1, c_1

of d_1 is changed. Considered the bit rotation number in step 2, the value of $d_{1,i+7}$ is changed. If carry occurs in $d_{1,i+7}$, upper bits of $d_{1,i+7}$ are changed, and this may result in breaking conditions in the upper bits of $d_{1,i+7}$. Therefore, if $d_{1,i+7} = 0$, we choose '+', and if $d_{1,i+7} = 1$, we choose '−'.

Basically, corrections are done from lower bit in order to avoid undesirable carry. However, there are some exceptions. From the expression for a_5, it can be said that $m_{1,31}$ is relevant to $a_{5,4}$, whereas, according to Table 2, $m_{1,27}$ is changed when $a_{5,32}$ is corrected. Therefore, if $a_{5,32}$ is corrected and the carry is transmitted from $m_{1,27}$ to $m_{1,31}$, the condition of $a_{5,4}$ is broken. To avoid this, we correct conditions in the following order: $i = 16, 18, 32, 4$. However, when $a_{5,4}$ is corrected and the carry is transmitted from $a_{5,4}$ to $a_{5,16}$, the condition on $a_{5,16}$ is broken. To avoid this, we set extra conditions $b_{4,4} = 1$ and $d_{1,11} = 0$. By setting $b_{4,4} = 1$, the value of $a_{5,4}$ becomes always 0 when $a_{5,4}$ is corrected because of a condition $a_{5,4} = b_{4,4}$. $d_{1,11} = 0$ guarantees that the carry never occurs in step 2 when $a_{5,4}$ is corrected.

Corrections for $d_{5,i}(i = 18, 30)$

d_5 is calculated by the following expression.

$$d_5 = a_5 + (d_4 + \phi(a_5, b_4, c_4) + m_6 + t) \lll 9$$

i-th bit of d_5 can be corrected by the message modification shown in Table 3.

Two extra conditions are set in Table 3. The purpose of the extra condition $d_{4,i-9} = 1$ is ignoring the change of c_4 in ϕ_{17}. The purpose of

表 6.3: The message modification for correcting "$d_{5,i}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
15	17	$m_{14} \leftarrow m_{14} \pm 2^{i-27}$	$c_4[\pm(i-9)], d_4, a_4, b_4$	
16	22	$m_{15} \leftarrow ((b_4 - c_4^{new}) \ggg 22) - b_4 - \phi(c_4^{new}, d_4, a_4) - t$	$b_4, c_4[\pm(i-9)], d_4, a_4$	
17	5		$a_5, b_4, c_4[\pm(i-9)], d_4$	$d_{4,i-9} = 1$
18	9		$d_5[\pm i], a_5, b_4, c_4[\pm(i-9)]$	$a_{5,i-9} \neq b_{4,i-9}$

the extra condition $a_{5,i-9} \neq b_{4,i-9}$ is reflecting the change of c_4 in ϕ_{18} in order to make change in $d_{5,i}$.

A Correction for $d_{5,32}$

From the expression of calculating d_5, if $m_{6,23}$ is changed, $d_{5,32}$ is corrected. Table 4 shows how to cancel the effect of the change of $m_{6,23}$ in the first round.

表 6.4: The message modification for correcting "$d_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
3	17	$m_2 \leftarrow m_2 + 2^6$	$c_1[23], d_1, a_1, b_0$	$c_{1,23} = 0$
4	22	$b_1 \leftarrow b_1 + 2^{22}$ $m_3 \leftarrow ((b_1^{new} - c_1^{new}) \ggg 22) - b_0 - \phi(c_1^{new}, d_1, a_1) - t$	$b_1[23], c_1[23], d_1, a_1$	
5	7	$m_4 \leftarrow ((a_2 - b_1^{new}) \ggg 7) - a_1 - \phi(b_1^{new}, c_1^{new}, d_1) - t$	$a_2, b_1[23], c_1[23], d_1$	
6	12	$m_5 \leftarrow ((d_2 - a_1) \ggg 12) - d_1 - \phi(a_2, b_1^{new}, c_1^{new}) - t$	$d_2, a_2, b_1[23], c_1[23]$	
7	17	$m_6 \leftarrow m_6 - 2^{22}$	$c_2, d_2, a_2, b_1[23]$	$d_{2,23} = 1$
8	22	$m_7 \leftarrow ((b_2 - c_2) \ggg 22) - b_1 - \phi(c_2, d_2, a_2) - t$	b_2, c_2, d_2, a_2	

In Table 4, the purpose of setting $c_{1,23} = 0$ is limiting the direction of change. Since a sufficient condition $b_{1,23} = c_{1,23}$ exists, we have to correct the value of $b_{1,23}$ after we modify m_2.

Correction for Other Conditions

Corrections for other conditions are shown in Table 5 to Table 11.

表 6.5: The message modification for correcting "$c_{5,18}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
10	12	$m_9 \leftarrow m_9 + 2^{23}$	$d_3[4], a_3, b_2, c_2$	$d_{3,4} = 0$
11	17	$m_{10} \leftarrow ((c_3 - d_3^{new}) \ggg 17) - c_2 - \phi(d_3^{new}, a_3, b_2) - t$	$c_3, d_3[4], a_3, b_2$	
12	22	$m_{11} \leftarrow m_{11} - 2^3$	$b_3, c_3, d_3[4], a_3$	$c_{3,4} = 1$
13	7	$m_{12} \leftarrow ((a_4 - b_3) \ggg 7) - a_3 - \phi(b_3, c_3, d_3^{new}) - t$	$a_4, b_3, c_3, d_3[4]$	
14	12	$m_{13} \leftarrow ((d_4 - a_4) \ggg 12) - d_3^{new} - \phi(a_4, b_3, c_3) - t$	d_4, a_4, b_3, c_3	

6.4.3 Estimation of the efficiency of corrections

Corrections mentioned in this section are probabilistic. We experimentally confirmed that all of these 14 conditions are satisfied after the message modifications with probability about 1/2.

Remark
We have found many other modification techniques for the second block (details are listed in appendix). By applying these modications, 12 conditions in the second round for the second block can be corrected. Therefore, the complexity of the attack for the second block becomes 2^{25}.

6.5 Modification Techniques for Shorten Repetition

In the method of Wang et al, if all of the sufficient condition are not satisfied after messages are modified, the algorithm goes back to step 15, and restart from randomly choosing m_{14} and m_{15}. Therefore, whenever a message does not satisfy all of the sufficient condition, single-message modification for m_{14} and m_{15}, multi-message modification and MD5 calculation for step 15 to step 64 are calculated. This complexity is big. In our research, we have found that it would be possible to go back to step 25 rather than step 15. This saves us the complexity of single-message modification, multi-message modification and calculation for step 15 to 24. By our modification, we estimated that the complexity becomes roughly 5/8 compared to the algorithm of Wang et al. In this section, we explain how

to restart the algorithm from step 25, and the estimation of the effect.

The calculation of step 25 is as follows,

$$a_7 = b_6 + (a_6 + \phi(b_6, c_6, d_6) + m_9 + t) \lll 5$$

Therefore, we change several bits of m_9 and this results in randomly changing the value of latter chaining variables. When an i-th bit of m_9 is changed, if we modify messages as shown in Table 12, the effect of the change is cancelled.

The number of i, where we can set corresponding extra conditions, is 2. Therefore, we can restart the collision search algorithm from the 25-th step $2^2 - 1 (= 3)$ times for each message satisfying all conditions untill step 25. We have estimated that the total complexity from step 15 to step 64 including both message modification would be 81 steps, and the complexity from step 25 to step 64 including our modification technique would be 41 steps. Therefore, the complexity of the attack by Wang et al. is $81 \times 4 = 324$ steps, whereas, the complexity of our attack is $81 + 41 \times 3 = 204$ steps. This indicates that the complexity becomes roughly 5/8 by applying our modification technique.

Remark

This modifiation technique is more effective for the second block than the first block. The number of i, where we can set corresponding extra conditions, is 9. Therefore, we can restart the algorithm from the 25-th step $2^9 - 1$ times for each message satisfying all conditions untill step 25. The complexity for the second block will become less than half.

6.6 Conclusion

In this paper, we proposed the method to correct 14 conditions in the second round. These corrections are probabilistic. We experimentally confirmed that all of these conditions were satisfied after the message modifications with probability about 1/2. On the other hand, 29 conditions are remained uncorrectable. Therefore, the complexity of our attack is 2^{30}.

We also proposed the more efficient collision search algorithm than that

of Wang et al. We showed that the total complexity of the collision search algorithm would be reduced to roughly 5/8.
These techniques can be conbined. Our result is the best of all existing attacks to MD5.
Finally, we show a collision message generated by our proposed method in Table 13. The message (M_0, M_1) and (M'_0, M'_1) have the same hash value.

Appendix (*List of Multi-Message Modifications for the 2nd block*)

表 6.6: The message modification for correcting "$c_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
13	7	$m_{12} \leftarrow m_{12} + 2^5$	$a_4[13], b_3, c_3, d_3$	$a_{4,13} = 0$
14	12	$m_{13} \leftarrow ((d_4 - a_4^{new}) \ggg 12) - d_3 - \phi(a_4^{new}, b_3, c_3) - t$	$d_4, a_4[13], b_3, c_3$	
15	17	$m_{14} \leftarrow ((c_4 - d_4) \ggg 17) - c_3 - \phi(d_4, a_4^{new}, b_3) - t$	$c_4, d_4, a_4[13], b_3$	
16	22	$b_4 \leftarrow b_4 - 2^{17} + 2^{22}$ $m_{15} \leftarrow ((b_4^{new} - c_4) \ggg 22) - b_3 - \phi(c_4, d_4, a_4^{new}) - t$	$b_4[-18, 23], c_4, d_4, a_4[13]$	$b_{4,18} = 1, b_{4,23} = 0$
17	5		$a_5, b_4[-18, 23], c_4, d_4$	$d_{4,23} = 0, (d_{4,18} = 1)$
18	9		$d_5, a_5, b_4[-18, 23], c_4$	$c_{4,18} = 1, c_{4,23} = 1$
19	14		$c_5[-32, 5], d_5, a_5, b_4[-18, 23]$	

表 6.7: The message modification for correcting "$b_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
13	7	$m_{12} \leftarrow m_{12} + 2^{31}$	$a_4[7], b_3, c_3, d_3$	$a_{4,7} = 0$
14	12	$m_{13} \leftarrow ((d_4 - a_4^{new}) \ggg 12) - d_3 - \phi(a_4^{new}, b_3, c_3) - t$	$d_4, a_4[7], b_3, c_3$	
15	17	$m_{14} \leftarrow ((c_4 - d_4) \ggg 17) - c_3 - \phi(d_4, a_4^{new}, b_3) - t$	$c_4, d_4, a_4[7], b_3$	
16	22	$b_4 \leftarrow b_4 - 2^{11}$ $m_{15} \leftarrow ((b_4^{new} - c_4) \ggg 22) - b_3 - \phi(c_4, d_4, a_4^{new}) - t$	$b_4[-12], c_4, d_4, a_4[7]$	$b_{4,12} = 1$
17	5		$a_5, b_4[-12], c_4, d_4$	$d_{4,12} = 0$
18	9		$d_5, a_5, b_4[-12], c_4$	$c_{4,12} = 1$
19	14		$c_5, d_5, a_5, b_4[-12]$	$d_{5,12} = a_{5,12}$
20	20		$b_5[-32], c_5, d_5, a_5$	

表 6.8: The message modification for correcting "$a_{6,18}$"

step	shift	Modify m_i	Chaining Variables		Extra Conditions
13	7	$m_{12} \leftarrow m_{12} - 2^{25}$	$a_4[-1], b_3, c_3, d_3$		$a_{4,1} = 1$
14	12	$m_{13} \leftarrow ((d_4 - a_4^{new}) \ggg 12) - d_3 - \phi(a_4^{new}, b_3, c_3) - t$	$d_4, a_4[-1], b_3, c_3$		
15	17	$m_{14} \leftarrow ((c_4 - d_4) \ggg 17) - c_3 - \phi(d_4, a_4^{new}, b_3) - t$	$c_4, d_4, a_4[-1], b_3$		
16	22	$m_{15} \leftarrow ((b_4 - c_4) \ggg 22) - b_3 - \phi(c_4, d_4, a_4^{new}) - t$	$b_4, c_4, d_4, a_4[-1]$		
17,2	5,12	$m_1 \leftarrow m_1 + 1$	a_5, b_4, c_4, d_4	$d_1[13], a_1, b_0, c_0$	$d_{1,13} = 0$
3	17	$m_2 \leftarrow ((c_1 - d_1^{new}) \ggg 17) - c_0 - \phi(d_1^{new}, a_1, b_0) - t$		$c_1, d_1[13], a_1, b_0$	
4	22	$m_3 \leftarrow ((b_1 - c_1) \ggg 22) - b_0 - \phi(c_1, d_1^{new}, a_1) - t$		$b_1, c_1, d_1[13], a_1$	
5	7	$m_4 \leftarrow ((a_2 - b_1) \ggg 7) - a_1 - \phi(b_1, c_1, d_1^{new}) - t$		$a_2, b_1, c_1, d_1[13]$	
6	12	$m_5 \leftarrow m_5 - 2^{12}$		d_2, a_2, b_1, c_1	

表 6.9: The message modification for correcting "$a_{6,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
4	22	$m_3 \leftarrow m_3 + 2^4$	$b_1[27], c_1, d_1, a_1$	$b_{1,27} = 0$
5	7	$m_4 \leftarrow ((a_2 - b_1^{new}) \ggg 7) - a_1 - \phi(b_1^{new}, c_1, d_1) - t$	$a_2, b_1[27], c_1, d_1$	
6	12	$m_5 \leftarrow m_5 - 2^{26}$	$d_2, a_2, b_1[27], c_1$	$a_{2,27} = 1$
7	17		$c_2, d_2, a_2, b_1[27]$	$(d_{2,27} = a_{2,27})$
8	22	$m_7 \leftarrow m_7 - 2^{26}$	b_2, c_2, d_2, a_2	

表 6.10: The message modification for correcting "$d_{6,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
9	7	$m_8 \leftarrow m_8 + 2^{15}$	$a_3[23], b_2, c_2, d_2$	$a_{3,23} = 0$
10	12	$m_9 \leftarrow ((d_3 - a_3^{new}) \ggg 12) - d_2 - \phi(a_3^{new}, b_2, c_2) - t$	$d_3, a_3[23], b_2, c_2$	
11	17	$m_{10} \leftarrow m_{10} - 2^{22}$	$c_3, d_3, a_3[23], b_2$	$d_{3,23} = 1$
12	22		$b_3, c_3, d_3, a_3[23]$	$c_{3,23} = 1$
13	7	$m_{12} \leftarrow ((a_4 - b_3) \ggg 9) - a_3^{new} - \phi(b_3, c_3, d_3) - t$	a_4, b_3, c_3, d_3	

表 6.11: The message modification for correcting "$c_{6,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
11	17	$m_{10} \leftarrow m_{10} + 2^{12}$	$c_3[30], d_3, a_3, b_2$	$c_{3,30} = 0$
12	22	$m_{11} \leftarrow m_{11} - 2^7$	$b_3, c_3[30], d_3, a_3$	$d_{3,30} = a_{3,30}$
13	7	$m_{12} \leftarrow ((a_4 - b_3) \ggg 7) - a_3 - \phi(b_3, c_3^{new}, d_3) - t$	$a_4, b_3, c_3[30], d_3$	
14	12	$m_{13} \leftarrow ((d_4 - a_4) \ggg 12) - d_3 - \phi(a_4, b_3, c_3^{new}) - t$	$d_4, a_4, b_3, c_3[30]$	
15	17	$m_{14} \leftarrow m_{14} + 2^{22} - 2^{29}$	$c_4[8], d_4, a_4, b_3$	$c_{4,8} = 0$
16	22	$m_{15} \leftarrow m_{15} - 2^7 - 2^{17}$	$b_4, c_4[8], d_4, a_4$	$(a_{4,8} = 0), (d_{4,8} = 1)$
17	5		$a_5, b_4, c_4[8], d_4$	$(d_{4,8} = 1)$
18	9		$d_5, a_5, b_4, c_4[8]$	$a_{5,8} = b_{4,8}$
19	14	($m_{11} \leftarrow m_{11} - 2^7$: modified in step 12)	c_5, d_5, a_5, b_4	

表 6.12: The message modification for shorten repetition

step	shift	Modify m_i	Chaining Variables	Extra Conditions
9	7	$m_8 \leftarrow m_8 + 2^{i+4}$	$a_3[i+12], b_2, c_2, d_2$	$a_{3,i+12} = 0$
10	12	$m_9 \leftarrow m_9 - 2^{i-1}$	$d_3, a_3[i+12], b_2, c_2$	$b_{2,i+12} = c_{2,i+12}$
11	17		$c_3, d_3, a_3[i+12], b_2$	$d_{3,i+12} = 0$
12	22		$b_3, c_3, d_3, a_3[i+12]$	$c_{3,i+12} = 1$
13	7	$m_{12} \leftarrow ((a_4 - b_3) \ggg 7) - a_3^{new} - \phi(b_3, c_3, d_3) - t$	a_4, b_3, c_3, d_3	

表 6.13: Generated collision messages in our proposed attack

M_0	0xcfdcc99611b8fc4b9dcf1099bab3b0f2c1d51f7112e9d1dc2110fa3e9f01eea5 06332cb1e0f307f88f6cf5ef9fb00f66a65fe4dbf50ed81f553b6443bc59b6e2
M_1	0xd958e84d2f5d1b9b53a46fce7ba577da94ac52f6ddc5506ae72e090ca18cf8cc f37eeff5085695806a77fb8a3e65b89590032d1d9513e57a7d283757ea659e11
M'_0	0xcfdcc99611b8fc4b9dcf1099bab3b0f241d51f7112e9d1dc2110fa3e9f01eea5 06332cb1e0f307f88f6cf5ef9fb08f66a65fe4dbf50ed81fd53b6443bc59b6e2
M'_1	0xd958e84d2f5d1b9b53a46fce7ba577da14ac52f6ddc5506ae72e090ca18cf8cc f37eeff5085695806a77fb8a3e65389590032d1d9513e57afd283757ea659e11
Hash value	0x1afe687725129c5fa5d52829e9bd5080

表 6.14: The message modification for correcting "$a_{5,i}(i = 4, 16, 18)$"

step	shift	Modify m_i	Chaining Variables
2	12	$m_1 \leftarrow m_1 \pm 2^{i-6}$	$d_1[\pm(i+7)], a_1, b_0, c_0$
3	17	$m_2 \leftarrow ((c_1 - d_1^{new}) \ggg 17) - c_0 - \phi(d_1^{new}, a_1, b_0) - t$	$c_1, d_1[\pm(i+7)], a_1, b_0$
4	22	$m_3 \leftarrow ((b_1 - c_1) \ggg 22) - b_0 - \phi(c_1, d_1^{new}, a_1) - t$	$b_1, c_1, d_1[\pm(i+7)], a_1$
5	7	$m_4 \leftarrow ((a_2 - b_1) \ggg 7) - a_1 - \phi(b_1, c_1, d_1^{new}) - t$	$a_2, b_1, c_1, d_1[\pm(i+7)]$
6	12	$m_5 \leftarrow ((d_2 - a_2) \ggg 12) - d_1^{new} - \phi(a_2, b_1, c_1) - t$	d_2, a_2, b_1, c_1

表 6.15: The message modification for correcting "$a_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
16	22	$m_{15} \leftarrow m_{15} - 2^0 - 2^8$	$b_4[-29,-31], c_4, d_4, a_4$	$b_{4,29} = 1, b_{4,31} = 1$
17 2	5 12	$m_1 \leftarrow m_1 + 2^{30}$	$a_5[\pm 32], b_4[-29,-31], c_4, d_4$ $d_1[11], a_1, b_0, c_0$	$a_{5,29} = 0, a_{5,30} = 0, (d_{4,29} = 0)$ $d_{1,11} = 0$
3	17	$m_2 \leftarrow ((c_1 - d_1^{new}) \ggg 17) - c_0 - \phi(d_1^{new}, a_1, b_0) - t$	$c_1, d_1[11], a_1, b_0, c_0$	
4	22	$m_3 \leftarrow ((b_1 - c_1) \ggg 22) - b_0 - \phi(c_1, d_1^{new}, a_1) - t$	$b_1, c_1, d_1[11], a_1$	
5	7	$m_4 \leftarrow ((a_2 - b_1) \ggg 7) - a_1 - \phi(b_1, c_1, d_1^{new}) - t$	$a_2, b_1, c_1, d_1[11]$	
6	12	$m_5 \leftarrow ((d_2 - a_2) \ggg 12) - d_1^{new} - \phi(a_2, b_1, c_1) - t$	d_2, a_2, b_1, c_1	

表 6.16: The message modification for correcting "$d_{5,18}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
14	12	$m_{13} \leftarrow m_{13} + 2^{28}$	$d_4[9], a_4, b_3, c_3$	$d_{4,9} = 0$
15	17	$m_{14} \leftarrow ((c_4 - d_4) \ggg 17) - c_3 - \phi(d_4, a_4^{new}, b_3) - t$	$c_4, d_4[9], a_4, b_3$	
16	22		$b_4, c_4, d_4[9], a_4$	$c_{4,9} = 0$
17	5		$a_5, b_4, c_4, d_4[9]$	$b_{4,9} = c_{4,9}$
18	9		$d_5[\pm 18], a_5, b_4, c_4$	

表 6.17: The message modification for correcting "$d_{5,30}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
7	17	$m_6 \leftarrow m_6 \pm 2^{20}$	$c_2[\pm 6], d_2, a_2, b_1$	
8	22	$m_7 \leftarrow ((b_2 - c_2) \ggg 22) - b_1 - \phi(c_2, d_2, a_2) - t$	$b_2, c_2[\pm 6], d_2, a_2$	
9	7	$m_8 \leftarrow ((a_3 - b_2) \ggg 7) - a_2 - \phi(b_2, c_2, d_2) - t$	$a_3, b_2, c_2[\pm 6], d_2$	
10	12	$m_9 \leftarrow ((d_3 - a_3^{new}) \ggg 12) - d_2 - \phi(a_3^{new}, b_2, c_2) - t$	$d_3, a_3, b_2, c_2[\pm 6]$	
11	17	$m_{10} \leftarrow ((c_3 - d_3^{new}) \ggg 17) - c_2 - \phi(d_3^{new}, a_3, b_2) - t$	c_3, d_3, a_3, b_2	

表 6.18: The message modification for correcting "$d_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
4	22	$m_3 \leftarrow m_3 + 1$	$b_1[23], c_1, d_1, a_1$	$b_{1,23} = 0$
5	7	$m_4 \leftarrow ((a_2 - b_1^{new}) \ggg 7) - a_1 - \phi(b_1^{new}, c_1, d_1) - t$	$a_2, b_1[23], c_1, d_1$	
6	12	$m_5 \leftarrow ((d_2 - a_2) \ggg 12) - d_1^{new} - \phi(a_2, b_1, c_1) - t$	$d_2, a_2, b_1[23], c_1$	
7	17	$m_6 \leftarrow m_6 - 2^{22}$	$c_2, d_2, a_2, b_1[23]$	$d_{2,23} = 0$
8	22	$m_7 \leftarrow ((b_2 - c_2) \ggg 22) - b_1 - \phi(c_2, d_2, a_2) - t$	b_2, c_2, d_2, a_2	

表 6.19: The message modification for correcting "$c_{5,18}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
8	22	$m_7 \leftarrow m_7 + 2^{13}$	$b_2[4], c_2, d_2, a_2$	$b_{2,4} = 0$
9	7	$m_8 \leftarrow ((a_3 - b_2) \ggg 7) - a_2 - \phi(b_2, c_2, d_2) - t$	$a_3, b_2[4], c_2, d_2$	
10	12	$m_9 \leftarrow ((d_3 - a_3^{new}) \ggg 12) - d_2 - \phi(a_3^{new}, b_2, c_2) - t$	$d_3, a_3, b_2[4], c_2$	
11	17	$m_{10} \leftarrow ((c_3 - d_3^{new}) \ggg 17) - c_2 - \phi(d_3^{new}, a_3, b_2) - t$	$c_3, d_3, a_3, b_2[4]$	
12	22	$m_{11} \leftarrow m_{11} - 2^3$	b_3, c_3, d_3, a_3	

表 6.20: The message modification for correcting "$c_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
15	17	$m_{14} \leftarrow m_{14} + 1$	$c_4[18], d_4, a_4, b_3$	$c_{4,18} = 0$
16	22	$m_{15} \leftarrow m_{15} - 2^{27}$	$b_4, c_4[18], d_4, a_4$	$(a_{4,18} = 1), (d_{4,18} = 1)$
17	5		$a_5, b_4, c_4[18], d_4$	$(d_{4,18} = 1)$
18	9		$d_5, a_5, b_4, c_4[18]$	$b_{4,18} = 0, (a_{5,18} = 0)$
19	14		$c_5[\pm 32], d_5, a_5, b_4$	

表 6.21: The message modification for correcting "$b_{5,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
13	7	$m_{12} \leftarrow m_{12} - 2^{31}$	$a_4[-7], b_3, c_3, d_3$	$a_{4,7} = 1$
14	12	$m_{13} \leftarrow ((d_4 - a_4^{new}) \ggg 12) - d_3 - \phi(a_4^{new}, b_3, c_3) - t$	$d_4, a_4[-7], b_3, c_3$	
15	17	$m_{14} \leftarrow ((c_4 - d_4) \ggg 17) - c_3 - \phi(d_4, a_4^{new}, b_3) - t$	$c_4, d_4, a_4[-7], b_3$	
16	22	$m_{15} \leftarrow m_{15} + 2^{21}$	$b_4[12], c_4, d_4, a_4[-7]$	$c_{4,7} = 1, b_{4,12} = 0$
17	5		$a_5, b_4[12], c_4, d_4$	$d_{4,12} = 0$
18	9		$d_5, a_5, b_4[12], c_4$	$c_{4,12} = 1$
19	14		$c_5, d_5, a_5, b_4[12]$	$d_{5,12} = a_{5,12}$
20	20		$b_5[\pm 32], c_5, d_5, a_5$	

表 6.22: The message modification for correcting "$a_{6,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
13	7	$m_{12} \leftarrow m_{12} - 2^6$	$a_4[22], b_3, c_3, d_3$	$a_{4,22} = 0$
14	12	$m_{13} \leftarrow ((d_4 - a_4^{new}) \ggg 12) - d_3 - \phi(a_4^{new}, b_3, c_3) - t$	$d_4, a_4[22], b_3, c_3$	
15	17	$m_{14} \leftarrow ((c_4 - d_4) \ggg 17) - c_3 - \phi(d_4, a_4^{new}, b_3) - t$	$c_4, d_4, a_4[22], b_3$	
16	22		$b_4, c_4, d_4, a_4[22]$	$c_{4,22} = 1$
17	5		$a_5[27], b_4, c_4, d_4$	$a_{5,27} = 0$
18	9	$m_6 \leftarrow m_6 - 2^{17} - 2^{26}$	$d_5, a_5[27], b_4, c_4$	
19	14		$c_5, d_5, a_5[27], b_4$	$b_{4,27} = 1$
20	20		$b_5, c_5, d_5, a_5[27]$	$c_{5,27} = d_{5,27}$
21	5		$a_6[\pm 32], b_5, c_5, d_5$	

表 6.23: The message modification for correcting "$d_{6,32}$"

step	shift	Modify m_i	Chaining Variables	Extra Conditions
9	7	$m_8 \leftarrow m_8 + 2^{15}$	$a_3[23], b_2, c_2, d_2$	$a_{3,23} = 0$
10	12	$m_9 \leftarrow ((d_3 - a_3^{new}) \ggg 12) - d_2 - \phi(a_3^{new}, b_2, c_2) - t$	$d_3, a_3[23], b_2, c_2$	
11	17	$m_{10} \leftarrow m_{10} - 2^{22}$	$c_3, d_3, a_3[23], b_2$	$d_{3,23} = 1$
12	22		$b_3, c_3, d_3, a_3[23]$	$c_{3,23} = 1$
13	7	$m_{12} \leftarrow ((a_4 - b_3) \ggg 9) - a_3^{new} - \phi(b_3, c_3, d_3) - t$	a_4, b_3, c_3, d_3	

第7章 How to Construct Sufficient Condition in Searching Collisions of MD5

Abstract

In Eurocrypt 2005, Wang et al. presented a collision attak on MD5. In their paper, they intoduced "Sufficient Condition" which would be needed to generate collisions. In this paper, we explain how to construct sufficent conditions of MD5 when a differential path is given. By applying our algorithm to a collision path given by Wang et al, we found that sufficient conditions introduced by them contained some unnecessary conditions. Generally speaking, when a differential path is given, corresponding sets of sufficient conditions is not unique. In our research, we analyzed the differential path found by Wang et al, and we found a different set of sufficient conditions from that of Wang et al. We have generated collisions by using our sifficient conditions.

7.1 Introduction

MD5 is a hash function proposed by Rivest in 1992 [13]. In Eurocrypt 2005, Wang et al. presented a collision attack on MD5 [10]. This attack found collisions of MD5 with complexity 2^{37}. This attack is very efficient compared to the complexity of an attack based on birthday paradox (2^{64}). However, the message differential used in this attack is given without any reasoning explanation. In their attack, they first construct sufficient

conditions which are needed to generate collisions. Then, they propose message modification techniques in order to satisfy these conditions.

Therefore, the procedure to begin a collision attack is as follows.

Step 1: Find message differentials and a differential path which generate collisions with high probability.

Step 2: Construct Sufficient Conditions which guarantees that desirable differential is always calculated.

Step 3: Find message modifications which can satisfy sufficient conditions with high probability.

In this paper, we explain how to work on Step 2, that is, how to construct sufficient conditions when a differential path is given. So far, a few papers have tried to analyze the sufficient condition table given by Wang et al. [16], [18]. However, these papers did not explain the systematic method to construct sufficient conditions. In this paper, we propose an algorithm to construct sufficient conditions. This paper does not mention Step 1 and Step 3. So far, no paper has been published about Step 1. In SCIS 2006, Yajima et al. publishes a paper which explains how to start the analysis on Step 1 [19]. Whereas, some papers have tried to improve Step 3 [15], [17], [28].

7.2 Discription of MD5

MD5 has the Markle/Damgard structure. MD5 compresses an arbitrary length messge into an 128-bit hash value. An input message is divided into 512-bit message blocks $(M_0, \ldots M_{n-1})$. First, the output of compression function H_1 is calculated by M_0 and H_0. Here H_0 is defined as follows:

$$H_0 = (0x67452301, 0xefcdab89, 0x98badcfe, 0x10325476)$$

The process of calculating H_1 is called the 1st block. Second, H_2 is calculated by M_1 and H_1. This process is called the 2nd block. Similarly, compression function is calculated until the last message M_{n-1} is used. Let H_n be the hash value of the message.

Compression Function

The input of compression funciton is a 512-bit message M_j. At the first, M_j is divided into 32-bit messages $(m_0, \ldots m_{15})$. The compression function consists of 64 steps. Step 1 to 16 are called 1st round, step 17 to 32 are called 2nd round, step 33 to 48 are called 3rd round, and step 49 to 64 are called 4th round. In each step, one of chaining varibals $a_i, d_i, c_i, b_i (1 \leq i \leq 16)$ is updated in this order. The figure of the calculation for each step is shown in Figure 1.

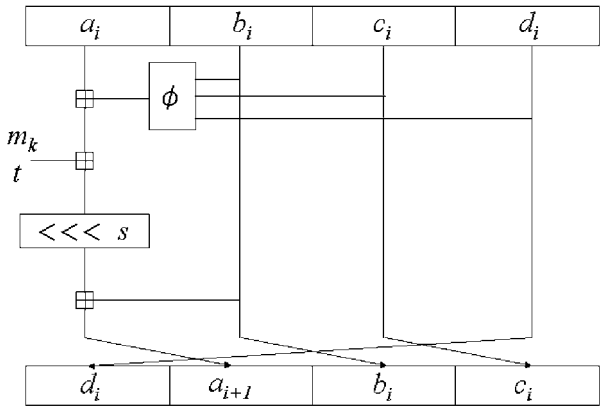

図 7.1: Structure of the Compuression Function

In Figure 1, s denotes a left cyclic shift number defined in each step. t denotes a constant defined in each step. m denotes a messge, and which m is used is defined in each round. ϕ is a non-linear boolean function defined in each round. Details of ϕ is as follows.

- 1st round: $\phi(X, Y, Z) = (X \wedge Y) \vee (\neg X \wedge Z)$
- 2nd round: $\phi(X, Y, Z) = (X \wedge Z) \vee (Y \wedge \neg Z)$
- 3rd round: $\phi(X, Y, Z) = X \oplus Y \oplus Z$
- 4th round: $\phi(X, Y, Z) = (X \vee \neg Z) \oplus Y$

After 64 steps are calculated, update initial values for the next message

block as follows.

$$aa_0 \leftarrow a_{16} + a_0$$
$$dd_0 \leftarrow d_{16} + d_0$$
$$cc_0 \leftarrow c_{16} + c_0$$
$$bb_0 \leftarrow b_{16} + b_0$$

At the last, let H_{j+1} be a concatination of these values.

$$H_{j+1} \leftarrow (aa_0|bb_0|cc_0|dd_0)$$

7.3 Sufficient Condition Construct Algorithm (SC algorithm)

Sufficient condition is constructed in order to guarantee that desirable differential is always calculated. Therefore, differential path search algorithm should be executed before the SC algorithm. In this paper, we assume that diffrentials of chaining variables for all steps are given in advance. If you want to get the detaild information for differential path search algorithm, please refer to [19].

The SC algorithm is going backward from the last step of the compression function. Therefore, if given collision differenatials are 2-block differentials, we start the SC algorithm from step 64 in the 2nd block.

7.3.1 Notation

To explain the SC algorithm, we need to explain notations. Let x_i be one of chaining variables a_i, b_i, c_i, d_i. Then, $x_{i,j}(1 \leq j \leq 32)$ represents the value of the j-th bit of chaining variable x_i. $x_i[j]$ and $x_i[-j]$ represent differentials of the j-th bit in chaining variable x_i. $x_i[j]$ means that differential of the j-th bit in x_i is 1, that is, $x'_{i,j} - x_{i,j} = 1$. Whereas, $x_i[-j]$ means that differential of the j-th bit in x_i is -1, that is, $x'_{i,j} - x_{i,j} = -1$.

In Figure 2, we name the differential after the rotation as Δu, we name the differential before the rotation as Δv, and we name the differential of the output of ϕ as $\Delta\phi$. Furthermore, $\Delta\phi_i$ denotes the $\Delta\phi$ in the i-th step, and $\Delta\phi_{i,j}$ denotes the j-th bit of $\Delta\phi_i$.

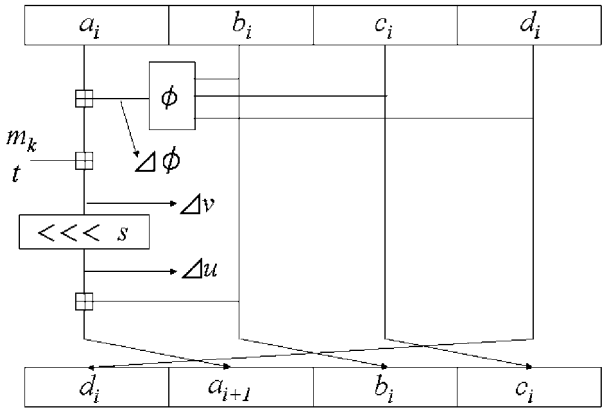

図 7.2: Notations

7.3.2 Calculation for $\Delta\phi$

We assume that differentials of all chaining variables are given in advance. Therefore, we assume that Δa_{i+1}, Δa_i, Δb_i, Δc_i and Δd_i in Figure 2 are given in advance. To control the behaviour of the boolean function ϕ, we first need to calculate ideal $\Delta\phi$ in the following way:

Step 1: Calculate $\Delta u = \Delta a_{i+1} - \Delta b_i$.

Step 2: Compute all possible Δv s.t. $\Delta u = (\Delta v \lll s)$.

Step 3: Calculate $\Delta\phi = \Delta v - \Delta m_k - \Delta a_i$.

In step 2, we cannot get all possible Δv by calculateing $\Delta v = (\Delta u \ggg s)$ because of the relationship between the effect of carry and rotation. Therefore, we should try all possible values of u, and calculate a value of $((u + \Delta u) \ggg s) - (u \ggg s)$ as a value of Δv. The value of u is a 32-bit number. Therefore, we need to repeat this calculation 2^{32} times for each step. This calculation is executed by a computer. The algorithm of this computation is as follows.

```
for (u=0x00000000 to 0xffffffff){
   v = u >>> s;
   u'= u + (delta u);
   v'= u'>>> s;
   diff = v' - v;
   counter[diff]++;
}
```

As a result of this computation, we will get at most 4 kinds of Δv. In order to raise the probability that a collision search algorithm succeeds in this step, we choose the most appeared value as a Δv. However, if sufficient conditions cannot be constructed for such Δv, it is possible to choose the differential which has smaller probability. The influence of the decrement of the probability by choosing such differential in the first round is ignoable. However, if it is in the second round or later, the influence of the decrement of the probability may be critical. In this case, choosing another differential path may be required.

7.3.3 Constructing Sufficient Conditions

Sufficient conditions can be classified in 2 types. One is conditions for controling the length of carry in chaining variabls, the other is conditions for controling the differentials of the output of the non-linear function ϕ. In this section, we explain how to construct each condition.

Conditions for Controling the Length of the Carry

These conditions are constructed in order to control the length of the carry in chaining variables. For example, we assume that a differential of a_2 is 2^6. In this case, if $a_{2,7}$ is 0, $a_{2,7}$ changes from 0 to 1, and no carry occurs by this differential. On the other hand, if $a_{2,7}$ is 1, the carry is transmitted to upper bits of a_2, and several bits may be changed by this differential. If the length of the carry is different, the result of moduler integer addition is not changed but the output of the non-linear function ϕ becomes different. Therefore, if we want to stop the differential from expanding, we construct sufficient conditions which prevent carry, in this case $a_{2,7} = 0$. On the other hand, if we want to expand the effect of the differential, we construct the sufficinent condition $a_{2,7} = 1$. Generally speaking, if the number of bits which will be changed by a differential is increased, the number of sufficient conditions where we need to construct is also increased. Therefore, we first construct conditions which stop the differenatial from expanding. Then, if these conditions are contradicted when we construct other conditions, we choose the other conditions which expands the effect of the differential.

Conditions for Controling the Output of the Non-linear Function ϕ

The output differential of the non-linear function ϕ can be controled by constructing appropreate sufficient conditions. To explain the basic idea of how to construct conditions, we give a small example. We assume following situations.

- The step is 64. (In this step, input chaining variables for ϕ are c_{16}, d_{16} and a_{16}.)
- The value of $\Delta\phi_{64}$ is $\pm 2^{31}$.
- Differentials of c_{16} is $c_{16}[-26, \pm 32]$.
- Differentials of d_{16} is $d_{16}[-26, \pm 32]$.
- Differentials of a_{16} is $a_{16}[\pm 32]$.

Under above situations, we first need to set $\Delta\phi_{64,26} = 0$. The value of $\Delta\phi_{64,26}$ is calculated depending on $c_{16,26}, d_{16,26}$ and $a_{16,26}$. Therefore, we consider all possible input values and find conditions of input variables whose output differential is 0. Table 1 shows the value of $\Delta\phi_{64,26}$ for all inputs. In Table 1, x, y, z represents the value of $c_{16,26}, d_{16,26}$,and $a_{16,26}$ respectively. The function ϕ is $\phi(x, y, z) = (x \vee -z) \oplus y$ since step 64 is in the 4th round. ϕ' is an output of ϕ after message diffrentials are added. Since a_{16} does not have differential in the 26-th bit, the value of z keeps unchanged. On the other hand, since c_{16} and d_{16} have differentials in the 26-th bit, the value of x and y are changed.

From the Table 1, we can get to know that a condition which sets $\Delta\phi_{64,26} = 0$ is "$z = 1$", that is, "$a_{16,26} = 1$".

Next, we need to set $\Delta\phi_{64,32} = \pm 1$. Similar to above procedure, we make the table of $\Delta\phi_{64,32}$ for all possible inputs, and extract a sufficient condition. Table 2 shows the table of $\Delta\phi_{64,32}$ for all possible inputs.

From the Table 2, we can get to know that a condition which sets the value of $\Delta\phi_{64,32} = \pm 1$ is "$x = z$", that is, "$c_{16,32} = a_{16,32}$".

表 7.1: Constructing Conditions for "$\Delta\phi_{64,26} = 0$"

x, y, z	$\phi(x, y, z)$	$\phi' = \phi(\neg x, \neg y, z)$	$\Delta\phi(= \phi' - \phi)$
0,0,0	1	0	-1
0,0,1	0	0	0
0,1,0	0	1	1
0,1,1	1	1	0
1,0,0	1	0	-1
1,0,1	1	1	0
1,1,0	0	1	1
1,1,1	0	0	0

表 7.2: Constructing Conditions for "$\Delta\phi_{64,32} = \pm 1$"

x, y, z	$\phi(x, y, z)$	$\phi' = \phi(\neg x, \neg y, \neg z)$	$\Delta\phi(= \phi' - \phi)$
0,0,0	1	0	-1
0,0,1	0	0	0
0,1,0	0	1	1
0,1,1	1	1	0
1,0,0	1	1	0
1,0,1	1	0	-1
1,1,0	0	0	0
1,1,1	0	1	1

7.3.4 Various Techniques to Avoid Contradiction

There are various techniques in order to avoid contradiction of the sufficient condition.

Extending Carry

We sometimes need to expand the effect of the differential by using a long carry. This situation occurs when $\Delta\phi_{i,j} \neq 0$ but j-th bit of all input chaining variables to ϕ_i are 0. As long as differentials of all input chaining variables are 0, it is impossible to make $\Delta\phi \neq 0$. Therefore, we extend the carry in the nearest lower bit until j-th bit. We give a small example to explain this.

Now, we assume the value of $\Delta\phi_i$ is 2^{-19}, and bit differentials of x_i, y_i, z_i which are input variables for the ϕ_i are $x_i[5], y_i[-16], z_i[10, 11, -12]$. Since a bit differential which is nearest to 2^{-19} is $y_i[-16]$, we expand the effect of $y_i[-16]$ until the 20th bit by using the carry. To archive this, we construct sufficient conditions $y_{i,16} = 0, y_{i,17} = 0, y_{i,18} = 0, y_{i,19} = 0, y_{i,20} = 1$.

Changing $\Delta\phi$

The value of $\Delta\phi$ can be changed by transforming expression. For example, $\Delta\phi = 2^{20}$ can be changed to $\Delta\phi = -2^{20}+2^{21}$, or $\Delta\phi = -2^{20}-2^{21}+2^{22}$ and so on. This transformation is useful in the following situation.

- Input chaining variables for ϕ_i are x, y and z.
- $\phi_i(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
- The value of $\Delta\phi_i$ is 2^{14}.
- Differentials of x is 0.
- Differentials of y is 0.
- Differentials of z is $z[-15, -16, -17. - 18, -19, 20]$.

In this case, it is impossible to make $\Delta\phi_{i,15} = 1$ by setting sufficient conditions. However, it is possible to make $\Delta\phi_{i,15} = -1$. Therefore, we replace the value of $\Delta\phi_i = 2^{14}$ with $\Delta\phi_i = -2^{14}-2^{15}-2^{16}-2^{17}-2^{18}+2^{19}$, and construct sufficient conditions $x_{15} = 0, \ldots, x_{20} = 0$ in order to make the ideal $\Delta\phi_i$.

7.3.5 Entire SC Algorithm

In this algorithm, whenever a new condition is constructed, we remember which step it is constructed. If a conditon is contradicted to another condition, we keep the newer condition, and then go back to the step where the older condition has been constructed. This concept is shown in Figure 3.

In Figure 3, we assume that we have two choices in step A, and we choose condition #1 at the first. Therefore, we keep condition #2 as a

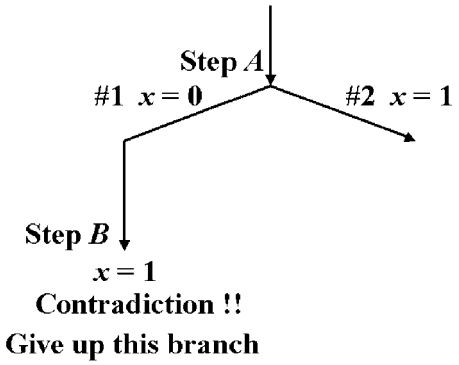

図 7.3: Concept of SC Algorithm

choice. However, if the chosen condition #1 becomes contradicted to other condition in step B, the algorithm goes back to step A, and chooses condition #2 in this time. Then the algorithm restarts from step A.

In the rest of this section, we explain details of the SC algorithm

Intialization Part

In all steps, calculate all possible $\Delta\phi$, then choose one which has the highest probability as $\Delta\phi$. Remember other candidates as choices (Discussed in 3.2).

Main Part

The SC algorithm goes backward from the last step of the last block to the 1st step of the 1st block. In each step, we construct sufficent conditions in order to stop the carry of chaining variables. Then we remember the condition which extends the carry as choices (Discussed in 3.3.1). After that, we construct conditions to control $\Delta\phi$. These conditions are constructed from lower bit, that is, from the 1st bit to the 32nd bit. In each bit, construct sufficient conditions in order to control the value of $\Delta\phi$ (Discussed in 3.3.2). If there are more than 1 choises of sufficient conditions for $\Delta\phi$, either of them can be chosen, then we remenber other conditions as choices. If conditions controling $\Delta\phi$ are contradicted or cannot be constructed, we have some choices. In order

to solve problems, we first try the process 1. If problems are not solved, we try the process 2 next. Then try the process 3, and finally, we try the process 4.

1. If $\Delta\phi \neq 0$ and all inputs to ϕ do not have a differential, we try to extend the nearest carry until this bit (Discussed in 3.4.1).

2. If $\Delta\phi$ can be changed, we try to change it (Discussed in 3.4.2).

3. If conditions controling $\Delta\phi$ are contradicted to other condition, we keep the new condition and go back to the step where the old condition has been constructed. Then choose other choices for the old condition (See Figure 3).

4. If all choices for old conditions are tried in process 3 and problems are still not solved, it is impossible to construct sufficient conditions for given $\Delta\phi$. Therefore, we choose another $\Delta\phi$ which we got in the Initialization part.

If the process 4 failed, that means no sufficient condition exists for the given differetial path.

7.4 Unnecessary Sufficient Conditions for the differential path of Wang et al.

To check the availability of the SC algirhtm, we apply this to the differential path given by Wang et al. Through this work, we found that a sufficient condition table given by them contained unnecessary conditions.

7.4.1 Unnecessary Conditions

So far, some papers have tried to analyze the mehtod of Wang et al, and pointed out mistakes of the sufficient condition table given by Wang et al. [16], [18]. Especially, [16] pointed out the lack of conditions in the 4th round, and claimed that the complexity estimation based on the sufficient condition table given by Wang et al. should be corrected. As

表 7.3: Generated Collision Messages by Using Our Sufficient Conditions

M_0	0x8b075d00f54501bce81f9cab86312f9d3a8bdca58446d56583e9e8365f99ddba 069badd582343c027f16e96793f95b7bdcdbe711c0dc183a6966bb7243c35a00
M_1	0x1e04541e498038f69d530565fafaf248484f373d4e37823ed76b4922c0b60954 fa3f9f9189df3b0e6307e1fad5ddcc4ff36210cafacc0b54f767ebf2b9391100
M'_0	0x8b075d00f54501bce81f9cab86312f9dba8bdca58446d56583e9e8365f99ddba 069badd582343c027f16e96793f9db7bdcdbe711c0dc183ae966bb7243c35a00
M'_1	0x1e04541e498038f69d530565fafaf248c84f373d4e37823ed76b4922c0b60954 fa3f9f9189df3b0e6307e1fad5dd4c4ff36210cafacc0b547767ebf2b9391100
Hash value	0x2e3757de930d2d15b1c77fa1ee924471

a result of applying our SC algorithm to the differential path of Wang et al, we also found the same mistakes.

Furthermore, we newly found unnecessary conditons. An unnecessary condition for the 2nd block is "$b_{16,26} = 1$", and an unnecessary condition for the 1st block is "$c_{3,31} = 0$". Removing "$b_{16,26} = 1$" is important since this condition is related to the estimation of the complexity. By removing "$b_{16,26} = 1$", the complexity of collision search algorithm for the 2nd block becomes 1/2.

7.4.2 Unnecessary Carry

Moreover, we found that the length of carry of a_2 for the 1st block could be shorten. In [10], they transmitted carry on the 7-th bit of a_2 until the 23rd bit, that is, the output of a_2 is $a_2[7, 8, \ldots 22, -23]$. However, bit differentials on $a_{2,21}, a_{2,22}$ and $a_{2,23}$ are not used so that we can stop the carry at the 20th step, that is, the output of a_2 becomes $a_2[7, 8, \ldots 19, -20]$. To archive this, we first change the condition on $a_{2,20}$ from "$a_{2,20} = 0$" to "$a_{2,20} = 1$" Then, we can remove sufficient conditions for $a_{2,21}, a_{2,22}, a_{2,23}$ and conditions controling output of the non-linear function ϕ on these bits. As a result, we can remove following conditions from the condition table of Wang et al. "$b_{1,21} = c_{1,21}$", "$b_{1,22} = c_{1,22}$", "$b_{1,23} = c_{1,23}$", "$a_{2,21} = 0$", "$a_{2,22} = 0$", "$a_{2,23} = 1$", "$d_{2,21} = 1$", "$d_{2,22} = 1$", "$d_{2,23} = 1$", "$c_{2,22} = 1$", "$c_{2,23} = 1$".

7.4.3 Merit of Removing Unnecessary Conditions

So far, we pointed out 12 unnecessary conditions in the 1st round, and removed them. However, the complexity of a collision search algorithm is not reduced and computing time of the algorithm is not changed, since these conditions could be corrected with very high probability by single-message modification written in [10]. However, they are still worth removing. In ISEC Nov 2005, an efficient collision search algorithm on MD5 was presented [17]. In this attack, multi-message modification using extra condition was proposed. If unnecessary conditions in the 1st block are removed, we can set extra conditions on those bits. It may result in making sufficient conditions in the 2nd round be correctable. Therefore, we should remove unnecessary conditions as soon as we find them.

7.5 Different Sufficient Conditions for the differential path of Wang et al.

In our analysis, we found that the number of sets of sufficient conditions which guarantee that given path is always calculated is not unique. There may exists more than 1. To explain this, we give a small exmaple. We assume following situations.

- Input chaining variables for ϕ_i are x, y and z.
- $\phi_i(x, y, z) = (x \wedge y) \vee (\neg x \wedge z)$
- The value of $\Delta\phi_i$ is 0.
- Differentials of x is $[\pm 32]$.
- Differentials of y is $[\pm 32]$.
- Differentials of z is 0.

In this case, we need to construct sufficient conditions which set $\Delta\phi_{i,32} = 0$. Table 4 is the table of $\Delta\phi_{i,32}$ for all possible inputs.

From the Table 4, it can be said that required conditions are "$x = 0, y \neq z$" or "$x = 1, y = z$". This is an example of the situation that we have

表 7.4: Constructing Conditions for "$\Delta\phi_{i,32} = 0$"

x, y, z	$\phi(x, y, z)$	$\phi' = \phi(\neg x, \neg y, z)$	$\Delta\phi(= \phi' - \phi)$
0,0,0	0	1	1
0,0,1	1	1	0
0,1,0	0	0	0
0,1,1	1	0	-1
1,0,0	0	0	0
1,0,1	0	1	1
1,1,0	1	0	-1
1,1,1	1	1	0

some choices to construct conditions. This fact indicates that it may be possible to construct sufficeint conditions which guarantee the differential given by Wang et al, but different from their condition table. In our research, we found such conditions for their differential, and try them instead of their sufficient conditions. In this computational experiment, conditions we changed are $d_{2,32}, a_{2,32}$, $a_{2,20}$ and $b_{1,32}$. We also removed unnecessary conditions where we explained in section 4. As a result, we could find a collision. We show a collision which was generated by using our sufficient conditions in Table 3.

7.6 Conclusion

In this paper, we proposed an algorithm for constructing sufficient conditions when a differential path is given. Then we applied this algorithm to the differential path given by Wang et al. [10]. As a result of this work, we found that 13 conditions can be removed from the table of the sufficient condition of Wang et al. One of the removed 13 conditions is a condition in the 4th round of the 2nd block. Therefore, the complexity of collision search algorithm for the 2nd block whose estimation of the complexity is based on the sufficient condition given by Want et al. becomes 1/2. Since [16] pointed out that [10] was lacked a condition in the 4th round of the 2nd block, by considering [16] and our result, total complexity becomes the same with [10].

We also searched a set of sufficient conditions which guarantees the differential path given by Wang et al. but different from their condition table. We changed 4 conditions in this work. At the last, we succeeded in generating a collision by using our sufficient conditions.

第8章　SHA-0に対する Message Modificationの考察

abstract

CRYPTO'05で，WangらはSHA-0に関してこれまで提案された方式よりもさらに効率的な攻撃を提案した．Wangらの方式を用いると2^{39}回のSHA-0演算でコリジョンメッセージを発見することができる．Wangらの方式はコリジョンメッセージを生成するための条件(sufficient condition)を全て満たすメッセージを探索することにより行われる．sufficient contidition を全て満たすメッセージがコリジョンメッセージとなる．彼らの方式ではステップ20までのsufficient conditionに対するmessage modificationは提案されているが，ステップ21以降のsufficient conditionに対するmessage modificationは提案されていない．そこで，本稿ではステップ21以降のsufficient conditionに対するmessage modificationを検討し，ステップ21，ステップ22のsufficient conditionを確率ほぼ1で修正できるmessage modificationを提案する．このmessage modificationを用いると計算量を2^{37}回のSHA-0演算でコリジョンを発見することができる．

8.1　はじめに

CRYPTO2005でWangらは，これまでに提案されたSHA-0に対するコリジョンアタック方式[2, 3, 4, 24, 25, 26]よりも効率的な方式を提案した[27]．この方式は算術演算を差分として用いた差分攻撃法であり，その計算量は2^{39}回のSHA-0演算である．Wangらの攻撃において最も重要な攻撃部品はローカルコリジョンである．ローカルコリジョンとは，任意のステップに差分を入力し，その差分を6ステップで打ち消す方式であ

る．このローカルコリジョンの開始位置を表現したものとして，Wang らはディスターバンスベクトルを定義している．さらに，ディスターバンスベクトルを基に，コリジョンメッセージを生成するための条件 (sufficient condition) を求めている．ここまでが準備段階である．具体的に，sufficient condition を求める手順は以下の通りである．

1. ディスターバンスベクトルを設定する．
2. ディスターバンスベクトルからメッセージ差分を求める．
3. メッセージ差分から全てのステップの内部変数の差分を求める．
4. 求めた内部変数の差分から sufficent condition を求める．

実際のコリジョンメッセージの生成には，message modification という技法が使われる．sufficient condition を満たしていない場合に，メッセージに修正を加えることにより，条件を満たすようにする技法である．具体的手順は以下の通りである．

1. ステップ 1 からステップ 14 の全ての sufficient condition を満たすメッセージ $(m_0, ..., m_{13})$ を生成する．
2. ステップ 15，ステップ 16 の全ての sufficient condition を満たすメッセージ (m_{14}, m_{15}) を生成する．
3. ステップ 17 からステップ 20 の全ての sufficient cnodition を満たすように message modification を行う．
4. 手順 1，2，3，で生成されたメッセージがステップ 21 以降の全ての sufficient condition を満たせばコリジョンメッセージとして出力し，満たさなければステップ 2 に戻る．

Wang らの方式は，ステップ 20 までの全ての sufficient condition に対しては message modification[20, 20, 21, 17, 9, 10] を用いることにより全て確率 1 で満すことができるが，21 ステップ目以降の sufficient condition に対する message modification の検討は一切行われていない，そのためステップ 21 以降の sufficient condition に対しては message modification を用いずに確率的に満たすようにしている．そこで，我々はステップ 21 以降の message modification について検討を行い，ステップ 21，ステップ 22 の sufficient condition を確率ほぼ 1 で満たす message modification を提案する．我々の message modification を用いることによりコリジョン探索の計算量が 2^{39} から 2^{37} に減少する．

8.2 SHA-0の構造

SHA-0は1993年にNISTによって提案されたハッシュ関数である[22]. SHA-0はcompression functionを繰り返し用いるMarkle-Damgard構造を持ち，任意長の入力に対して160ビットのデータを出力する．SHA-0のcompression functionの構造は以下の通りである．

Step 1. 入力メッセージMを32ビットずつ16個のメッセージ$m_0, m_1, ..., m_{15}$に分割する．

Step 2. m_{16}からm_{79}を以下のようなメッセージ拡大により計算する．

$$m_i = m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}$$

Step 3. ステップiの内部変数a_i, b_i, c_i, d_i, e_iを以下の手順により計算する．

$$\begin{aligned} a_i &= (a_{i-1} \lll 5) + f(b_{i-1}, c_{i-1}, d_{i-1}) + e_{i-1} \\ &\quad + m_{i-1} + k_{i-1} \\ b_i &= a_{i-1} \\ c_i &= b_{i-1} \lll 30 \\ d_i &= c_{i-1} \\ e_i &= d_{i-1} \end{aligned}$$

この手順を80回実行する．ステップ1からステップ20が1ラウンド目，ステップ21からステップ40が2ラウンド目，ステップ41からステップ60が3ラウンド目，ステップ61からステップ80が4ラウンド目になる．kは各ラウンドごとに定められた定数である．また，関数fの仕様は以下の通りである．

$$\begin{aligned} &1\text{ラウンド目} : (b \wedge c) \vee (\neg b \wedge d) \\ &2\text{ラウンド目} : b \oplus c \oplus d \\ &3\text{ラウンド目} : (b \wedge c) \vee (c \wedge d) \vee (d \wedge b) \\ &4\text{ラウンド目} : b \oplus c \oplus d \end{aligned}$$

Step 4. $(a_0+a_{80}, b_0+b_{80}, c_0+c_{80}, d_0+d_{80}, e_0+e_{80})$がcompression functionの出力になる．

8.3 WangらのSHA-0に対する攻撃手法[27]

本章ではWangらの攻撃手法において重要なローカルコリジョン，ディスターバンスベクトル，sufficient condition，message modificationについて説明する．

ローカルコリジョン

Wangらの攻撃に最も重要な攻撃部品はローカルコリジョンである．ローカルコリジョンとは表1のようにSHA-0の任意のステップに差分を入力し，その差分を6ステップで打ち消す方式である．このローカルコリジョンを用いて差分パスを作る．

step	Δm	Δa	Δb	Δc	Δd	Δe
i	2	2				
$i+1$	2^6		2			
$i+2$	2			2^{31}		
$i+3$	2^{31}				2^{31}	
$i+4$	2^{31}					2^{31}
$i+5$	2^{31}					

表 8.1: ローカルコリジョン

ディスターバンスベクトル

ローカルコリジョンを用いて差分パスを作る際，ローカルコリジョンのスタート位置に注目する．まずローカルコリジョンのスタート位置を示すベクトル $(x_{-5}, ..., x_{-1}, x_0,$
$..., x_{79})$ を用意する．このベクトルをディスターバンスベクトルと呼ぶ．ステップ i からローカルコリジョンがスタートする場合は $x_{i-1} = 1$，スタートしない場合は $x_{i-1} = 0$ と定義する．

ディスターバンスベクトルはローカルコリジョンの最初のメッセージ差分が入力される位置を示しているので，メッセージと同様に扱う．よって，ディスターバンスベクトルは $x_{-5}, ..., x_{15}$ が自由に設定でき，$x_{16}, ..., x_{79}$ はメッセージと同様に $x_i = x_{i-3} \oplus x_{i-8} \oplus x_{i-13} \oplus x_{i-16}$ によって計算される．また，差分パスを生成する条件として，ディスターバンスベクトルに以

下の条件が必要になる.

- $x_{75} = 0, ..., x_{79} = 0$
- $x_{11} = x_{-5} \oplus x_{-3} \oplus x_3 \oplus x_8$
- $x_{12} = x_{-4} \oplus x_{-2} \oplus x_4 \oplus x_9$
- $x_{13} = x_{-3} \oplus x_{-1} \oplus x_5 \oplus x_{10}$
- $x_{14} = x_{-2} \oplus x_0 \oplus x_6 \oplus x_{11}$
- $x_{15} = x_{-1} \oplus x_1 \oplus x_7 \oplus x_{12}$

この条件を基に Wang らは表 2 のディスターバンスベクトルを選択した.

i	value
$-5, ..., -1$	0 0 1 1 1
$0, ..., 19$	0 1 1 1 1 0 0 1 0 1 0 0 1 0 1 0 1 0 0 0
$20, ..., 39$	0 1 1 0 0 0 1 0 0 0 1 0 1 1 1 0 0 0 0 0
$40, ..., 59$	0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0
$60, ..., 79$	1 1 0 0 1 1 0 1 0 1 1 1 0 1 0 0 0 0 0 0

表 8.2: Wang らのディスターバンスベクトル

Sufficient Condition

sufficient condition はコリジョンメッセージを生成するための十分条件であり，この条件を全て満たすメッセージを生成することができれば，コリジョンメッセージを作ることができる．この sufficient condition はディスターバンスベクトルに注目し生成する.

$x_i = 1$(ただし 2 ラウンド目と 4 ラウンド目):

$$a_{i-1,4} = a_{i-2,4} \ (or \ a_{i-1,4} \neq a_{i-2,4})$$
$$a_{i,2} = m_{i-1,2}$$

$x_i = 1$(ただし 3 ラウンド目):

$$a_{i-1,4} \neq a_{i-2,4}$$
$$a_{i,2} = m_{i-1,2}$$
$$a_{i+1,32} \neq a_{i-1,2}$$
$$a_{i+2,32} \neq a_{i+1,2}$$

$x_i = 0$ の時は，ローカルコリジョンの発生は無いので，sufficient condition は考えなくてよい．また，1 ラウンド目の sufficient condition は本稿の議論には必要ないので省略する.

Message Modification

1ラウンド目のsufficient conditionを確率1で満たす方式としてmessage modificationがある.

- ステップ i $(1 \leq i \leq 16)$ のmessage modificaiton:

  1. sufficient conditionを満たす a_i を生成する.
  2. $m_{i-1} \leftarrow a_i - (a_{i-1} \lll 5) - f(b_{i-1}, c_{i-1}, d_{i-1})$
     $-e_{i-1} - k_{i-1}$

- ステップ i $(17 \leq i \leq 20)$ のmessage modification:
  (例: $a_{17,32} = 0$ の修正)

  1. $m_{15} \leftarrow m_{15} \pm 2^{26}$ を実行する.
  2. $a_{16} = (a_{15} \lll 5) + f(b_{15}, c_{15}, d_{15}) + e_{15} + m_{15} + k$ より $a_{16} \leftarrow a_{16} \pm 2^{26}$
  3. $a_{17} = (a_{16} \lll 5) + f(b_{16}, c_{16}, d_{16}) + e_{16} + m_{16} + k$ より $a_{17} \leftarrow a_{17} \pm 2^{31}$

コリジョン探索手順

以下の手順でコリジョン探索を行う.

Step 1: message modificationを用い，ステップ1からステップ14のsufficient conditionを満たすメッセージ $m_0, ..., m_{13}$ を生成する.

Step 2: message modificationを用い，ステップ15，ステップ16のsufficient conditionを満たすメッセージ m_{14}, m_{15} を生成する.

Step 3: message modificationを用い，ステップ17からステップ20のsufficient conditionを満たすメッセージ m_{15} を生成する.

Step 4: Step 1，2, 3によって求めたメッセージ M に対して，ステップ21以降のsufficient conditionを満たすかチェックする．満たさなければStep2に戻る.

Step 5: $M' = M + \Delta M$ を計算する．このとき，$\Delta M, M'$ がコリジョンメッセージになる.

Wang らの差分パスから上記の方法でコリジョン探索を行うと，計算量 2^{39} 回の SHA-0 演算でコリジョンメッセージを発見することができる．

8.4 新たな Message Modification の提案

Wang らの方式ではステップ 20 までの sufficient condition は message modification を用いることにより確率 1 で満たすことができる．しかし，Wang らの方式ではステップ 21 以降の sufficient condition に対して message modification を行っていない．そこで，ステップ 21 以降の sufficient condition に対する message modification について検討する．

8.4.1 ステップ 21 の sufficient condition について

ステップ 21 に sufficient condition $a_{21,4} = a_{20,4}$ (or $a_{21,4} \neq a_{20,4}$) が存在する．この sufficient condition は $m_{15} \leftarrow m_{15} \oplus 2^{10}$ を実行すると確率ほぼ 1 で修正される．次に，このメッセージ変更を行うと sufficient condition を修正できる理由を説明する．

ステップ 21 の sufficient condition に対する message modification の考察

m_{15} に 2^{10} の差分を入れると図 1 のように差分が伝播していく．ただし，図 1 の差分の伝播は桁上がりが無い場合の差分の伝播である．図中の四角で囲まれている差分は必ず出現する差分であり，それ以外の差分は関数 f の特徴から出現する可能性のある差分である．図 1 より，桁上がりが無い場合，m_{15} の差分 2^{10} が $a_{16} \rightarrow a_{17} \rightarrow a_{18} \rightarrow a_{19} \rightarrow a_{20} \rightarrow a_{21}$ を通って，a_{21} の 4 ビット目に確率 1 で影響を与える．

次に桁上がりがある状況を考える．この場合，a_{21} の 4 ビット目に影響を与え，この差分を打ち消す差分が発生する可能性がある．しかし，計算機実験からこの差分が発生する確率はきわめて低いことが確認できた．よって，m_{15} に 2^{10} の差分を入れると確率ほぼ 1 で a_{21} の 4 ビット目に影響を与えることができる．

図 8.1: message modification による差分の流れ

8.4.2 ステップ 22 の sufficient condition について

ステップ 22 の sufficient condition に対する message modificaion の考察を行う．ステップ 22 の sufficient condition は $a_{21,4} = a_{20,4}$ $(or\ a_{21,4} \neq a_{20,4})$ と $a_{22,2} = m_{21,2}$ である．

ステップ 17 で用いられるメッセージ m_{16} はメッセージ拡大によって生成されるが，仕様を変更し m_{16} をメッセージ拡大に左右されずに自由に変更可能な状況を考える．$m_{16} \leftarrow m_{16} \oplus 2^i$ と変更し，$a_{21,4} = a_{20,4}$ $(or\ a_{21,4} \neq a_{20,4})$ と $a_{22,2} = m_{21,2}$ を修正できる i を探索した．$i = 8$ の時 $a_{22,2} = m_{21,2}$ を確率ほぼ 1 で満たすことを計算機実験により確認できた．$i = 10$ の時 $a_{21,4} = a_{20,4}$ $(or\ a_{21,4} \neq a_{20,4})$ を確率ほぼ 1 で満たすことを計算機実験により確認できた．

次に，この状況を実際の SHA-0 の構造に対して実現可能であることを説明する．

- **$a_{22,2} = m_{21,2}$ の修正**

まず，$a_{22,2} = m_{21,2}$ を修正する方法について検討する．ステップ 17 の計算は以下の通りである．

$$a_{17} = (a_{16} \lll 5) + f(b_{16}, c_{16}, d_{16}) + e_{16} + m_{16} + k_{16}$$

ここで，$m_{16} \leftarrow m_{16} \oplus 2^8$ を実行すると，$a_{17} \leftarrow a_{17} \pm 2^8$ となる．すなわち，m_{16} を修正する代わりに $a_{16}, b_{16}, c_{16}, e_{16}$ のどれかに差分を立てることにより，$a_{17} \leftarrow a_{17} \pm 2^8$ が実現できれば，$m_{16} \leftarrow m_{16} \oplus 2^8$ と同じ状況を実現することができる．$e_{16} \leftarrow e_{16} \oplus 2^8$ によって，$a_{17} \leftarrow a_{17} \pm 2^8$ を実現する方法を考える．$e_{16} \leftarrow e_{16} \oplus 2^8$ は SHA-0 の構造より $a_{12} \leftarrow a_{12} \oplus 2^{10}$ であれば実現することができる．$a_{12} = (a_{11} \lll 5) + f(b_{11}, c_{11}, d_{11}) + e_{11} + m_{11} + k_{11}$ なので，$m_{11} \leftarrow m_{11} \oplus 2^{10}$ を実行すれば $a_{12} \leftarrow a_{12} \oplus 2^{10}$ を実現することができる．このとき a_{12} に差分による桁上がりが起きないように，extra condition として $a_{12,11} = m_{11,11}$ を追加しておく．次に，ステップ 17 までの計算で必要な差分以外に余分な差分が発生しないように条件を設定またはメッセージに差分を入れる．

ステップ 13:

ステップ 13 では以下の計算を行う．

$$a_{13} = (a_{12} \lll 5) + f(b_{12}, c_{12}, d_{12}) + e_{12} + m_{12} + k_{12}$$

ここで，$a_{12} \leftarrow a_{12} \oplus 2^{10}$ は，a_{13} に影響を与える．この影響を打ち消すために，m_{12} に差分を入力すれば a_{13} への影響を打ち消すことができる．$m_{12} \leftarrow m_{12} \oplus 2^{15}$ を実行し，extra condition として $m_{12,16} \neq m_{11,11}$ を設定しておけば a_{13} への影響を打ち消すことができる．

ステップ 14:

ステップ 14 では以下の計算を行う．

$$a_{14} = (a_{13} \lll 5) + f(b_{13}, c_{13}, d_{13}) + e_{13} + m_{13} + k_{13}$$

$b_{13} \leftarrow b_{13} \oplus 2^{10}$ が起こるので，この変更が a_{14} に影響しないように，extra condition として $c_{13,11} = d_{13,11}$ すなわち $a_{11,13} = a_{10,13}$ を設定しておく．この extra condition を設定すると，関数 f の特徴から差分を打ち消すことができる．

ステップ 15,16:

ステップ 15 で a_{15}，ステップ 16 では a_{16} を計算するので，ステップ 14 と同様に関数 f の特徴を用いて a_{15}，a_{16} への影響を打ち消す．すなわち，extra condition として $a_{13,9} = 0$，$a_{14,9} = 1$ を設定すれば a_{15}，a_{16} への影響を打ち消すことができる，

以上の考察より，表 3 のように extra condition を設定し，m_{11}，m_{12} に差分を入力すると，$m_{16} \leftarrow m_{16} \oplus 2^8$ と同じ状況を作ることができ，$a_{22,2} = m_{21,2}$ を確率ほぼ 1 で修正することができる．このときのステップ

Step	Modification	extra condition
12	$m_{11} \leftarrow m_{11} \oplus 2^{10}$ $a_{12} \leftarrow a_{12} \oplus 2^{10}$	$a_{12,11} = m_{11,11}$
13	$m_{12} \leftarrow m_{12} \oplus 2^{15}$	$m_{12,16} \neq m_{11,11}$
14		$a_{11,13} = a_{10,13}$
15		$a_{13,9} = 0$
16		$a_{14,9} = 1$

表 8.3: $a_{22,2} = m_{21,2}$ の修正

12 からステップ 17 までの差分の流れは図 2 の通りである．また，extra condition はステップ 16 以前の条件なので，計算量に影響を与えることは無い．

図 8.2: ステップ 12 からステップ 17 までの差分の流れ

- $a_{21,4} = a_{20,4}$ $(or\ a_{21,4} \neq a_{20,4})$ の修正

$a_{21,4} = a_{20,4}$ $(or\ a_{21,4} \neq a_{20,4})$ の修正は $a_{22,2} = m_{21,2}$ の修正と同様に表 4 のように条件を設定し，差分を入力すれば修正することができる．またこの修正でのステップ 12 からステップ 17 までの差分の流れは図 3 のようになる．

以上の考察より，我々の message modification を用いると，ステップ 22 の sufficient condition を確率ほぼ 1 で満たすことができる．

Step	Modification	extra condition
12	$m_{11} \leftarrow m_{11} \oplus 2^{12}$ $a_{12} \leftarrow a_{12} \oplus 2^{12}$	$a_{12,13} = m_{11,13}$
13	$m_{12} \leftarrow m_{12} \oplus 2^{17}$	$m_{12,18} \neq m_{11,13}$
14		$a_{11,15} = a_{10,15}$
15		$a_{13,11} = 0$
16		$a_{14,11} = 1$

表 8.4: $a_{21,4} = a_{20,4}$ $(or\ a_{21,4} \neq a_{20,4})$ の修正

図 8.3: ステップ 12 からステップ 17 までの差分の流れ

注 (ステップ 22 の message modification の失敗例)

ステップ 21 の message modification と同様に m_{15} を以下のように修正し，ステップ 22 の sufficient condition の修正可能性について実験を行った．

$$m_{15} \leftarrow m_{15} \oplus 2^i$$

その結果，どの i をとっても，ステップ 22 の sufficient condition を高い確率で修正可能な変更は存在しないことを確認した．また $m_0, ..., m_{14}$ に関して同様の修正を行いステップ 22 の sufficient condition の修正可能性について実験を行った．その結果，ステップ 22 の sufficient condition を高い確率で修正可能な変更は存在しないことを確認した．

8.4.3 計算量

今回提案した message modification はステップ 21 とステップ 22 の 3 つの sufficient condition を修正することができる．ステップ 22 までの Wang らの方式の計算量は約 80 ステップで，我々の message modification を用いると約 20 ステップである．よって，ステップ 22 までの計算量は 2^{-2} 減らすことができる．これが，全体の計算量に影響し，攻撃に必要な計算量は 2^{39} から 2^{37} に減少する．

8.5 まとめ

今回我々はステップ 21 の sufficient condition とステップ 22 の sufficient condition に対する message modification を考察し，それぞれの sufficient condition を確率ほぼ 1 で修正できる message modification を提案した．この message modification を用いると 2^{37} 回の SHA-0 演算でコリジョンメッセージを発見することができる．

付録 (SHA-1 について [28])

CRYPTO'05 で，Wang らは SHA-1 に対するコリジョンアタック手法を提案した．Wang らはこの手法を用いると 2^{69} 回の SHA-1 演算でコリジョンを発見できると主張している．しかし，Wang らの手法は理論上での検討しかされておらず，実際に 2^{69} でコリジョンを発見できるか検討されていない．そこで本章では Wang らの SHA-1 に対するコリジョンアタック手法を検討する．

SHA-1 の仕様は SHA-0 のメッセージ拡大を以下のように変更したものである [23].

$$m_i = (m_{i-3} \oplus m_{i-8} \oplus m_{i-14} \oplus m_{i-16}) \lll 1$$

SHA-1 へのコリジョンアタック手法は SHA-0 へのコリジョンアタック手法と同様にローカルコリジョンを用いる．まず，ローカルコリジョンのスタート位置を決定するディスターバンスベクトル $(x_0, ..., x_{79})$ を用意する．メッセージ拡大が SHA-0 と異なるので，ディスターバンスベクトル x_i $(i = 0, ..., 79)$ は 32 ビットになる．ディスターバンスベクトルの条件は SHA-0 のディスターバンスベクトルの条件と同じである．Wang ら

は2ブロックを用いることでディスターバンスベクトルの条件$x_{75}, ..., x_{79}$を取り除いた．これにより，総当り攻撃より高速にコリジョンメッセージを発見できる差分パスをWangらは求めている．しかし，Wangらの手法を検討すると，WangらのSHA-1に対するコリジョンアタック手法だけではコリジョン探索をするには不十分であることが分かった．

次に，Wangらの手法が不十分な理由を説明する．Wangらは1ブロック目の出力に関してsufficient conditionをつけなくても良いと主張している．Wangらの手法を用いた場合1ブロック目の出力差分Δh_{1block}は以下のようになる．そのため，出力値にsufficient conditionをつけない場合，差分の桁上がりをコントロールできない．

$$\Delta h_{1block} = (2^{15}, 2^7\ 2^8, 0, 2^8, 0)$$

1ブロック目の出力値は2ブロック目の初期値として用いられるので，コリジョン探索を実行するごとに2ブロック目の初期値の差分によって発生する2ブロック目の内部変数の差分をコントロールすることができない．また，2ブロック目の初期値は2ブロック目のステップ1からステップ4までの計算に用いられるので，2ブロック目のメッセージ差分から発生する差分をコントロールできない．よって，Wangの手法を用いるとコリジョン探索毎に2ブロック目の差分が変化する．すなわち，Wangらの手法を実現するためには，2ラウンド目の差分パスを2^{69}回以下のSHA-1演算で生成するアルゴリズムが必要になる．しかし，Wangらの論文では，2ブロック目の差分パスを生成する方法は示されていない．以上の理由から，Wangらの手法は，その記述のレベルではコリジョン探索を行うには不十分である．

今後の課題

Wangらの手法が実現可能性を検討するために，2ブロック目の差分パスを効率的に生成するアルゴリズムが存在するかを検討する必要がある．

第9章　まとめと今後の課題

本報告書では，WangのSHA-1に対する攻撃法を解説し，その問題点を指摘した．さらに，SHA-1よりも構造が単純なハッシュ関数MD4, MD5, SHA-0への攻撃を通して，攻撃技術の理解，習得を行なった．

来年度の研究課題として，SHA-1に対するWangの攻撃法の詳細な検討を行ない，だれでも検証が可能な形での記述を行なうことである．その上で，正確な計算量の評価を行ない，SHA-1の攻撃の現実社会への有効性を検証する予定である．さらに，可能な限りWangらの攻撃の効率化を行なうことも課題の一つである．本報告書3章および4章で示した項目を中心に検討を行なう予定である．

関連発表リスト

1. Y. Naito, Y. Sasaki, N. Kunihiro and K. Ohta, "Improved Collision Attack on MD4 with Probability Almost1," In 8th ICISC,pp.122-135, 2005.

2. Y. Sasaki, Y. Naito, N. Kunihiro and K. Ohta, "Improved Collision Attack on MD5,", ISEC2005-104, pp. 35-42, 2005 年 11 月.

3. Y. Sasaki, Y. Naito, J. Yajima, T. Shimoyama, N. Kunihiro and K. Ohta, 'MD5 のコリジョン探索における Sufficient Condition の導出法について," SCIS2006-4E1-1, 2006.

4. 内藤祐介，佐々木悠，下山武司，矢嶋純，國廣昇，太田和夫, "SHA-0 の Message Modification に関する考察," SCIS2006- 4E1-3, 2006.

関連図書

[1] B. den Boer, A. Bosselaers: Collisions for the compression function of MD5, EUROCRYPT'93, 1993

[2] E. Biham, R. Chen: Near collision of SHA-0, CRYPTO2004, LNCS 3152, pp290–305, 2004.

[3] E. Biham, R. Chen, A. Joux, P. Carribault: Collisions of SHA-0 and Reduced SHA-1, EUROCRYPT2005, LNCS3494, pp.36–57, 2005.

[4] F. Charband, A. Joux: Differential Collisions in SHA-0, CRYPTO'98, 1998

[5] H. Dobbertin: Cryptanalysis of MD4, First Software Encryption 1996, LNCS 1039, 1996

[6] H. Dobbertin: The First Two Rounds of MD4 are Not One-Way, Fast Software Encryption 1998, 1998.

[7] R. Rivest: The MD4 Message Digest Algorithm, CRYPTO'90 Proceedings, 1991, http://theory.lcs.mit.edu/~rivest/Rivest-MD4.txt

[8] X. Wang, D. Feng, X. Lai, H. Yu: Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD, rump session, CRYPTO 2004, e-Print, 2003.

[9] X. Wang, X. Lai, D. Feng, H. Chen, X. Yu: Cryptanalysis of the Hash Functions MD4 and RIPEMD, Advances in EUROCRYPT2005, LNCS 3494, pp. 1–18, 2005.

[10] X. Wang, H. Yu: How to break MD5 and Other Hash Functions, Advances in EUROCRYPT2005, LNCS 3494, pp. 19–35, 2005.

[11] H. Dobbertin: The status of MD5 after a recent attack, CryptoBytes 2 (2), 1996

[12] R. Rivest: The MD4 Message Digest Algorithm, CRYPTO'90 Proceedings, 1992, http://theory.lcs.mit.edu/~rivest/Rivest-MD4.txt

[13] R. Rivest: The MD5 Message Digest Algorithm, CRYPTO'90 Proceedings, 1992, http://theory.lcs.mit.edu/~rivest/Rivest-MD5.txt

[14] X. Wang, Y. Lisa Yin, H. Yu: Finding Collisions in the Full SHA-1, Crypto 2005, LNCS 3621, pp. 17-36, 2005

[15] V. Klima: Finding MD5 Collisions on a Notebook PC Using Multi-message Modifications, Cryptology ePrint Archive 102, April 2005, http://eprint.iacr.org/2005/102.pdf

[16] Jie Liang, Xuejia Lai: Improved Collision Attack on Hash Function MD5, Cryptology ePrint Archive 425, November 2005, http://eprint.iacr.org/2005/425.pdf

[17] Yu Sasaki, Yusuke Naito, Noboru Kunihiro, Kazuo Ohta: Improved Collision Attack on MD5, ISEC2005-104, pp. 35-42, 2005.

[18] Jun Yajima, Takeshi Shimoyama: On the collision search and the sufficient conditions of MD5, ISEC 2005-78, pp.15-22, 200,

[19] Jun Yajima, Takeshi Shimoyama, Yu Sasaki, Yusuke Naito, Noboru Kunihiro, Kazuo Ohta: How to construct a differential path of MD5 for collision search, SCIS 2006, 2006.

[20] V. Klima. *Finding MD5 Collisions - a Toy For a Notebook*. Cryptology ePrint archive, Report 2005/075.

[21] Y. Naito, Y. Sasaki, N. Kunihiro and K. Ohta. *Improved Collision Attack on MD4 with Probability Almost 1*. ICISC'05, pp122–135, Springer-Verlag, 2005.

[22] NIST. *Secure hash standard*. Federal Information Processing Stacdard, FIPS-180, May 1993.

[23] NIST. *Secure hash standard*. Federal Information Processing Stacdard, FIPS-180-1, April 1995.

[24] X. Wang, D. Feng, H. Chen, X. Lai and X. Yu. *Collision for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD*. In Rump Session of CRYPTO'04 and Cryptology ePrint Archive, Report 2004/199.

[25] X. Wang. *The Collision Attack on SHA-0*. In Chinese, to appear on www.infosec.edu.cn, 1997.

[26] X. Wang. *The Improved Collision Attack on SHA-0*. In Chinese, to appear on www.infosec.edu.cn, 1998.

[27] X. Wang, H. Yu and Y. Lisa Yin. *Efficient Collision Search Attack on SHA-0*. CRYPTO'05, LNCS 3621, pp1–16, Springer-Verlag, 2005.

[28] X. Wang, Y. Lisa Yin and H. Yu. *Finding Collisions in the Full SHA-1*. CRYPTO'05, LNCS 3621, pp17–36, Springer-Verlag, 2005.