計測自動制御学会東北支部 60 周年記念学術講演会 (2024.12.17)
資料番号 S60-18

ラウンド加算 DFA によるハッシュ関数 Tiger のメッセージ復元の検討

Consideration of Hash Function Tiger Message Recovery Using Raund Addtion DFA

○茂木大樹 *，吉川英機 *

○ Daiki Mogi*, Hideki Yoshikawa*

*東北学院大学大学院 工学研究科

*Graduate School of Engineering, Tohoku Gakuin University

キーワード： ハッシュ関数 (Hash Function)，サイドチャネル攻撃 (Side Channel Attack)，差分故障解析 (Differential Fault Anarysis)，メッセージ復元 (Message Recovery)，ラウンド加算 (Round Addition)

連絡先： 〒 984-8588　宮城県仙台市若林区清水小路 3–1 東北学院大学大学院 工学研究科 電気工学専攻
茂木大樹, Email: s236521009@g.tohoku-gakuin.ac.jp

1. はじめに

ハッシュ関数とは任意長のデータを入力として，ハッシュ値やダイジェストと呼ばれる固定長の値を出力する一方向性関数である．暗号学的ハッシュ関数は 3 つの特性を有する必要があり，同一のハッシュ値となる異なる入力を計算することが困難である「衝突困難性」，ハッシュ値から元の入力を計算することが困難である「現像計算困難性」，およびハッシュ値と入力のペアから同様のハッシュ値となる入力を計算することが困難である「第 2 現像計算困難性」である．ハッシュ関数はオンライン上のデータの送受信の際の改ざん検出に利用されるディジタル署名や情報サービスにおけるアカウントのパスワード保存など，情報セキュリティに関する仕組みに利用されている技術である．上記 3 つの特性のうち衝突困難性と現像計算困難性は特にハッシュ関数の安全性に関与する性質であり，対策が十分に講じられている必要がある．

ハッシュ関数に対する攻撃の一つにはハッシュ関数が実装されているシステムに対する外部からの攻撃であるサイドチャネル攻撃が存在している．その中でも，異常電圧の印加により命令スキップを生じさせたり，レーザーなどによるデータ破壊を行うことで攻撃を行う差分故障解析 (Differential Fault Analysis: 以下 DFA) と呼ばれるものがあり，MD5 や SHA-1，SHA-2 などの従来の汎用ハッシュ関数に対しても適用する研究が存在する [5)6)7)]．

本稿の手法であるラウンド加算 DFA は既に Feistel 構造のブロック暗号における秘密鍵導出に対して適用して，実際に秘密鍵が導出できることが示されている [1)]．またラウンド加算 DFA を利用したハッシュ関数の現像計算困難性については MD5，SHA-1，SHA-2 については復元できることが示されている [2) 3)]．

本稿ではMD5, SHA-1, SHA-2と同様にMerkle-Damgård構造を有するTigerに対して本手法を拡張し，メッセージ復元について検討し，結果を述べる．

2. ラウンド加算DFA

DFAは”Differential Fault Analysis”の略であり，意図的に故障を誘発させることで秘密情報の取得を行う攻撃法である．また特定のマイクロコントローラには規定よりも低い電圧を印加した際に命令がスキップされる現象が確認されている[4]．これを利用し，ラウンド加算DFAでは繰り返し処理のインクリメント命令をスキップさせることで余分な処理を行い異常出力を得ることで，正常出力との比較によって秘密情報を取得する．Algorithm1はラウンド加算DFAの例であり，コード内の「$i \leftarrow i+1$」に該当するインクリメント命令をスキップすることでラウンド処理「$Process()$」をもう一度実行させるようにする．

Algorithm 1 Encrypt Round Function

```
function ENCRYPT_ROUND_FUNCTION
    i ← 0
    while i < n do
        Process()
        i ← i + 1 // Skip Point
    end while
end function
```

3. ハッシュ関数Tigerに対するラウンド加算DFAの概要

3.1 ハッシュ関数Tigerの概要

Tigerは1996年に”Ross Anderson”と”Eli Biham”によって発表されたMD4，MD5，SHA1の代替となるように設計されたハッシュ関数である[4]．64bit単位で処理を行い，192bitのハッシュ値を出力する．Tigerの構造は，入力されたデータに対しパディングを行いメッセージブロックを生成し，メッセージブロックを処理部分に入力し，これを繰り返すMerkle–Damgård構造となっている．Tigerにはパディング時に0x01を付加するもので，0x80を付加する際にはTiger2と区別される．

Algorithm 2 Tiger: Hash Function

```
function TIGER(H, X, n)
    O ← H
    for i = 0, ··· , n − 1 do
        O ← PASS(O, X[i], 5)
        X[i] ← KeySchedule(X[i])
        O ← PASS(O, X[i], 7)
        X[i] ← KeySchedule(X[i])
        O ← PASS(O, X[i], 9)
        O ← Final(O, H)
        H ← O
    end for
    return O
end function
```

Tigerでは Algorithm2で示すような処理を行っている．Xはメッセージブロックであり，1つにつき512bitのデータが格納されている．処理単位の64bitに合わせ，

$$X = \{x_0, x_1, \cdots, x_7\}$$

と64bitのデータが8つ並んだ構造となっている．nはメッセージブロックの数を表している．

Hは192bitの内部状態であり，3つの64bitデータが並んだ構造である．仕様では$IV_0 = \{a_0, b_0, c_0\}$とするとき，

$$\begin{cases} a_0 = \text{0x0123456789ABCDEF} \\ b_0 = \text{0xFEDCBA9876543210} \\ c_0 = \text{0xF096A5B4C3B2E187} \end{cases}$$

であり，これが初期の内部状態となる．メッセージブロックの数だけ処理を繰り返し，最終の内部状態がハッシュ値として出力される．Fig.1はAlgorithm2のメッセージブロック1つ分の処理図となる．

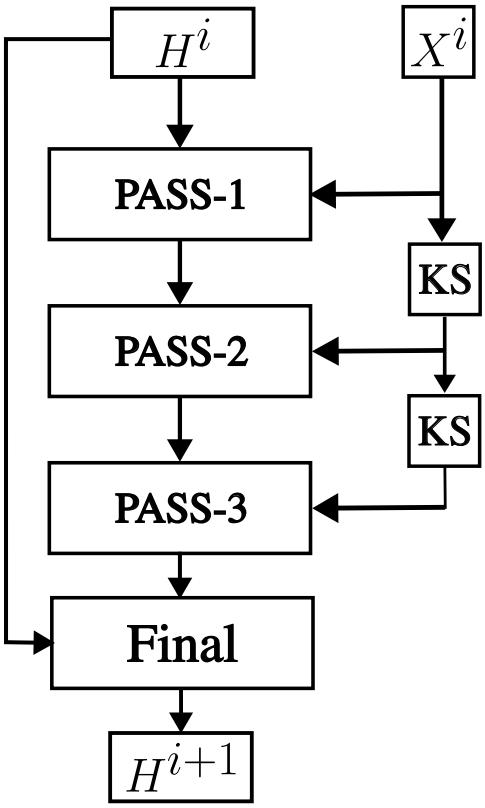

Fig. 1　Tiger の構造図．(KS:KeySchedule)

関数 PASS() は Algorithm3 に示す操作を行う．これはメッセージブロック内の値 8 つをそれぞれ入力していき，内部状態を変化させている．これを図示したものが Fig.1 である．

Algorithm 3 PASS()

function PASS(H, X, mul)
  for $i = 0, \cdots, 7$ **do**
    $H \leftarrow round(H, x[i], mul)$
  end for
  return H
end function

関数 round() は Algorithm4 の操作を行う．ここでの加算，減算は 2^{64} を法とし $2^{64}-1$ を上回れば 0 へ戻り，0 を下回れば $2^{64}-1$ となる演算である．$\oplus$ は排他的論理和 (XOR) を表す．乗算は 64bit を超える上位 bit を切り捨てて処理される．even() と odd() は c の偶数バイト，奇数バイトの値から変形した値である．mul は 3 回の PASS の処理ごとに 5,7,9 と代入される．これを図示したものが Fig.2 である．

Algorithm 4 round()

function ROUND(H, x, mul)
  $a \leftarrow H[0]$
  $b \leftarrow H[1]$
  $c \leftarrow H[2]$
  $c \leftarrow c \oplus x$
  $a \leftarrow a - even(c)$
  $b \leftarrow b + odd(c)$
  $b \leftarrow b * mul$
  $H[0] \leftarrow b$
  $H[1] \leftarrow c$
  $H[2] \leftarrow a$
  return H
end function

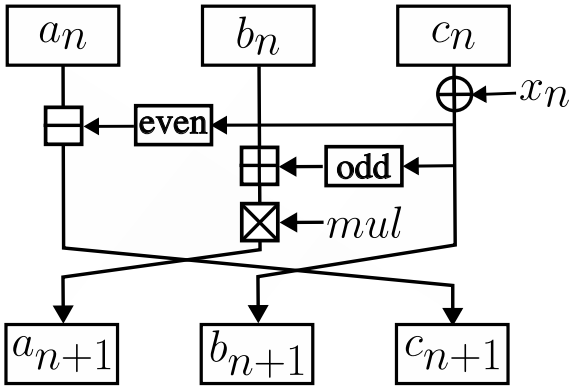

Fig. 2　ラウンド処理の構造図

これらの処理後，メッセージブロックは KeySchedule によって変化し，再度 mul の値を変えて PASS の処理を行う．KeySchedule() は Algorithm5 で示す．

3 回の PASS の後，内部状態は処理前の内部状態との処理 Final() を通し，その回の最終状態とする．Algorithm6 は Final() の処理を示す．

3.2　ラウンド加算 DFA によるメッセージスケジュールの導出

Tiger に対するラウンド加算 DFA を適用し，実際にメッセージスケジュールを導出する．便宜上 PASS1$\tilde{3}$ における $\{x_0, x_1, \cdots, x_7\}$ を通しで

Algorithm 5 KeySchedule()

function KEYSCHEDULE(X)
$x[0] \leftarrow x[0] - (x[7] \oplus$ 0xA5A5A5A5A5A5A5A5)
$x[1] \leftarrow x[1] \oplus x[0]$
$x[2] \leftarrow x[2] + x[1]$
$x[3] \leftarrow x[3] - (x[2] \oplus ((\neg x[1]) << 19))$
$x[4] \leftarrow x[4] \oplus x[3]$
$x[5] \leftarrow x[5] + x[4]$
$x[6] \leftarrow x[6] - (x[5] \oplus ((\neg x[4]) >> 23))$
$x[7] \leftarrow x[7] \oplus x[6]$
$x[0] \leftarrow x[0] + x[7]$
$x[1] \leftarrow x[1] - (x[0] \oplus ((\neg x[7]) << 19))$
$x[2] \leftarrow x[2] \oplus x[1]$
$x[3] \leftarrow x[3] + x[2]$
$x[4] \leftarrow x[4] - (x[3] \oplus ((\neg x[2]) >> 23))$
$x[5] \leftarrow x[5] \oplus x[4]$
$x[6] \leftarrow x[6] + x[5]$
$x[7] \leftarrow x[7] - (x[6] \oplus$ 0x0123456789ABCDEF)
return X
end function

Algorithm 6 Final()

function FINAL(Ho, Hi)
$Ho[0] \leftarrow Ho[0] \oplus Hi[0]$
$Ho[1] \leftarrow Ho[1] - Hi[1]$
$Ho[2] \leftarrow Ho[2] + Hi[2]$
return Ho
end function

$\{x_0, x_1, \cdots, x_{23}\}$ と表記する．スキップするポイントは x_{22} が使用されたラウンドであり，出力によって得られた正常ハッシュ値を O，異常ハッシュ値を O' とする．Fig.3 はラウンド加算 DFA が適用された際の内部状態の流れを示している．元のメッセージブロックが 1 つである場合最終的な内部状態は定められた初期値を用いて処理されハッシュ値としているため，最終ラウンドは以下の式で導出できる．

$$\begin{cases} a_{24} = O[0] \oplus H_0[0] \\ b_{24} = O[1] + H_0[1] \\ c_{24} = O[2] - H_0[2] \end{cases}$$

異常出力も同様であり，異常ハッシュ値を O' とすると，

$$\begin{cases} a'_{24} = O'[0] \oplus H_0[0] \\ b'_{24} = O'[1] + H_0[1] \\ c'_{24} = O'[2] - H_0[2] \end{cases}$$

となる．

処理内の各状態を変形していくと，(1) の式となる．

$$\begin{aligned} a_{23} &= c_{24} + even(b_{24}) \\ a'_{23} &= c'_{24} + even(b'_{24}) \end{aligned} \tag{1}$$

b_{23} と b'_{23} は乗算によって上位ビットが切り捨てられているため不明になっているが，以下の Algorithm7 によって確定させる．

Algorithm 7 state b search

for $i = 0, \cdots, 10$ **do**
$tmp \leftarrow \{((i << 64) \oplus a_{24})/9\} - odd(b_{24})$
if $\{(tmp + odd(b_{24})) * 9\} == a24$ **then**
$b_{23} \leftarrow tmp$
end if
end for

b_{23} と b'_{23} を探索したら，異常出力の c'_{23} に着目する．c'_{23} は導出した b'_{23} と正常出力から導出した a_{23} より求めることができる．導出式は式 (2) に示す．

$$c'_{23} = a_{23} - even(b'_{23}) \tag{2}$$

ここでメッセージスケジュール x_{23} は $b'_{24} = x_{23} \oplus c'_{23}$ より式 (3) となる．

$$x_{23} = c'_{23} \oplus b'_{24} \tag{3}$$

x_{23} から c_{23} を求めることができ，

$$c_{23} = b_{24} \oplus x_{23} \tag{4}$$

x_{22} は異常出力の状態から，式 (5) で求められる．

$$x_{22} = c_{23} \oplus b'_{23} \tag{5}$$

以上の工程で x_{22}, x_{23} を導出でき，これらを利用し正常出力の状態 22 までを導出することが可能となる．他のポイントでも DFA を行うことでさらに内部状態とメッセージスケジュールを辿ることができる．

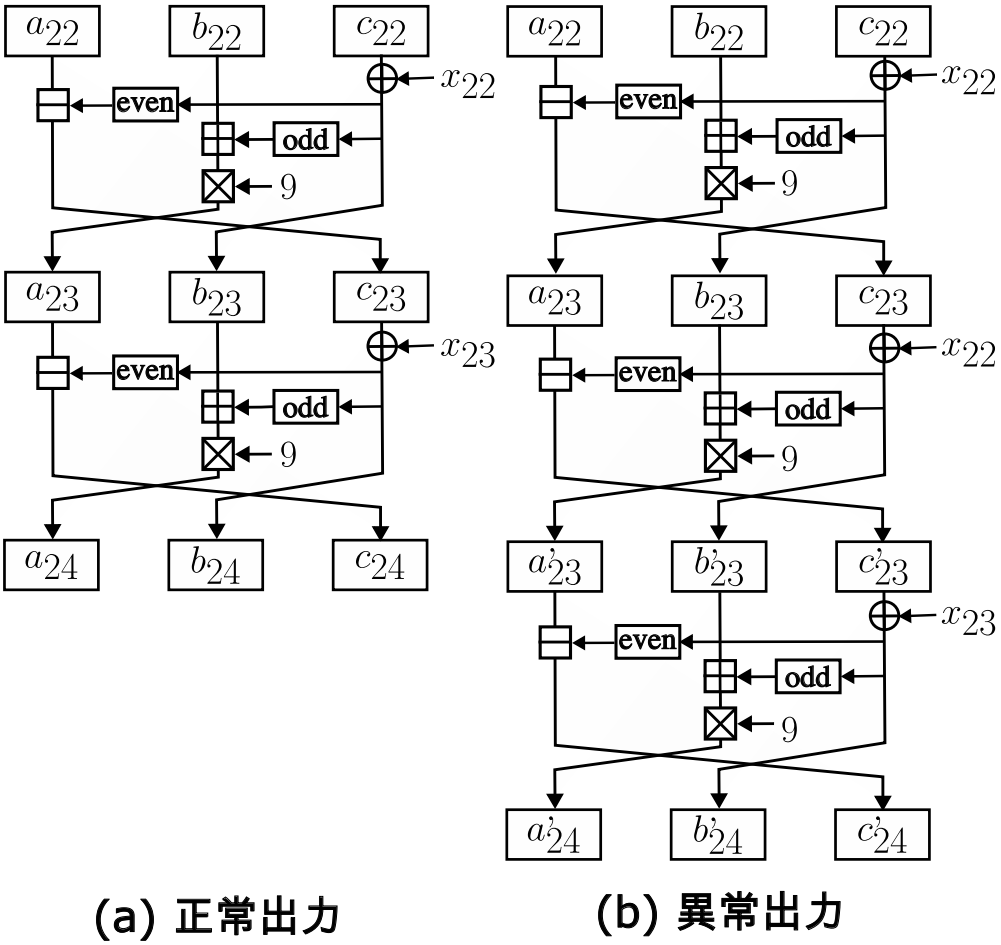

Fig. 3　正常出力と異常出力

3.3　メッセージスケジュールよるメッセージ復元

Algorithm5 の KeySchedule によってメッセージスケジュールが生成されることから，この処理を逆に辿ることでメッセージの復元ができる．復元にはラウンド 16~23 で使用された 8 つのメッセージスケジュールがあればよい．

4.　おわりに

本稿ではハッシュ関数 Tiger に対するラウンド加算 DFA の適用によるメッセージ復元について述べた．その結果，1 回のラウンド加算 DFA によって 2 つのメッセージスケジュールを得ることができ，このことから SHA-2 と同様に 4 回のラウンド加算 DFA を行うことでメッセージを復元することができた．なお本手法では処理の最終状態を得る必要があり，メッセージブロックが 2 つのときには不明となるため適さない．よってメッセージブロックが 1 つ，つまりは入力が 383bit のとき，4 回のラウンド加算 DFA を適用することでメッセージ復元が可能となる．今後の研究課題としては，他構造を含めたハッシュ関数への適用を検討する．

参考文献

1) 吉川英機: ラウンド加算 DFA による Feistel 型ブロック暗号における鍵導出法，電子情報通信学会論文誌, Vol.J101–D No.7, 1027/1036 (2018)

2) 伊藤久晃，吉川英機: ラウンド加算 DFA

を用いたハッシュ関数への攻撃に関する一報告, 情報理論とその応用シンポジウム (SITA2022) ポスター発表, November 29 (2022)

3) 茂木大樹, 神永正博, 吉川英機: ラウンド加算 DFA によるハッシュ関数 SHA2 の現像復元の検討, 2024 年度 電気関係学会 東北支部連合大会, August 29, 2024.

4) Masahiro Kaminaga, Arimitsu Shikoda, Hideki Yoshikawa: Development and evaluation of a microstep DFA vulnerabillity estimation method, IEICE Electronics Express, Vol.8, No.22, 1899/1904 (2011)

5) R. Anderson, E. Biham: Tiger: A Fast New Hash Funtion (1996)

6) Wei Li, Zhi Tao, Dawu Gu, Yi Wang, Zhiqiang Liu, Ya Liu: Differential Fault Analysis on the MD5 Compression Function, JOURNAL OF COMPUTERS, VOL. 8, NO.11 November (2013)

7) Ludger Hemme, Lars Hoffmann: Differentail Fault Analysis on the SHA1 Compression Function, 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (2011)

8) Kazuki Nakamura, Koji Hori, Shoichi Hirose: Algebraic Fault Analysis of SHA-256 Compression Function and Its Application, Information 2021, 12, 433. https://doi.org/10.3390/info12100433 (2021)