IPA®
INFORMATION-TECHNOLOGY PROMOTION AGENCY, JAPAN

算術演算をベースとする
ハッシュ関数安全性評価手法に関する調査
調査報告書

2008 年 5 月

独立行政法人　情報処理推進機構

本書の目的

本報告書は、SHA-1 等の算術演算ベースのハッシュ関数に対する安全性評価の手法を調査し、算術演算をベースとするハッシュ関数の評価ツールの作成を目的とした基礎検討を行うことを目指す。

本書が対象とする読者

本報告書が対象とする読者としては、情報セキュリティに関する一般的知識を持ち、ハッシュ関数の安全性に関して興味がある技術者を想定している。そのため一般的と思われる情報セキュリティ用語（ハッシュ関数、暗号解読等）については本書内において、用語の説明を省略する場合がある。

本書の構成

本書における各章の関係は**図　i** の通りである。

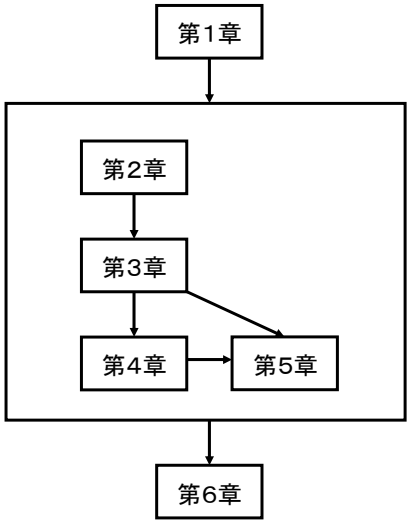

図　i．本書の構成

1. はじめに

近年、暗号プリミティブの解析技術が著しく進歩するに従い、DES 暗号や一部のハッシュ関数に関してはその利用に対し注意が必要となっている。特に、電子政府システム、電子商取引システムなどにおけるデジタル署名に用いられる SHA-1 と呼ばれるハッシュ関数に関しては、「衝突」の危険性が増している。とりわけハッシュ関数 SHA-1 の安全性に対しては、米国標準技術研究所（NIST：National Institute of Standards and Technology）でも注意喚起を行っており、次世代ハッシュ関数 AHS（SHA-3)の公募が計画されている。

次世代のハッシュ関数としては、従来のハッシュ関数とは異なる安全性の根拠に基づいて構成される可能性もあるが､現用の SHA-1 や MD5 といった算術演算をベースとしたもの、あるいは AES の設計思想に基づく S-box をベースとしたもの等、これまでに知られている安全性根拠に基づいて構成される可能性も高い。特に現在広く用いられている SHA-1 や MD5 といった算術演算をベースとしたハッシュ関数の安全性評価手法については、より一層緻密な評価技術の開発が期待されている。

そこで、本稿では、次世代ハッシュ関数の安全性を評価するための手法として、現在研究開発が進められているハッシュ関数の動向を調査し、特に算術演算に基づくハッシュ関数の安全性評価手法をツール化した際の、適用可能性、実現性、課題を抽出し報告する。

1.1. ハッシュ関数の安全性

ハッシュ関数として必要とされる安全性指標としては、次の 3 つが知られている。

- 一方向性（Pre-image Resistance）
  y が与えられたとき，y = Hash(x) となる x を求めることが困難であること
- 第二原像困難性（2nd Pre-image Resistance）
  x が与えられたとき，Hash(x) = Hash(x')（$x' \neq x$) となる x' を求めることが困難であること
- 衝突困難性（Collision Resistance）
  Hash(x) = Hash(x') となる異なる x, x' を求めることが困難であること

これらの中で、衝突困難性を満たせば、他の二つも満たすことが知られていることから、本稿では主としてハッシュ関数の衝突困難性に対する安全性評価に関する考察を進めることとする。

衝突困難性を評価する上では、その評価基準としてバースデーパラドックスに基づく攻撃（バースデーアタックとも呼ばれることがある）に必要な計算量を比較対象として用いる。この値と、知られている最良の攻撃法で必要な計算量や計算資源とを比較し、バース

デーパラドックスに基づく攻撃よりも少ない計算コストとなった場合、そのハッシュ関数は、暗号理論的には安全でないとされている。

定理 1.

バースデーパラドックス

t ビット出力のハッシュ関数のコリジョンを確率 p で見つけるのに必要なハッシュ関数の計算回数を k 回とすると、 k は以下の式から求められる。

$$k = \frac{1+\sqrt{1+2^{t+3}\log\left(\frac{1}{p-1}\right)}}{2} = O\left(2^{t/2}\right)$$

1.2. 用語説明

本節では、本稿で用いる用語についてまとめる。

算術演算をベースとしたハッシュ関数

算術演算をベースとしたハッシュ関数における圧縮関数は、一般的に図 1 のような構造をしている。入力されたメッセージは、ブロック長の倍数のサイズになるようにパディングされ、先頭から順次圧縮関数に入力される。各ブロック内部では、入力されたブロック長サイズのメッセージ M をさらに複数のデータに分割し、一部のステップにおいてはそのまま処理に使用する。残りのステップについては「メッセージ拡大」と呼ばれる処理によって得られた拡大メッセージが処理に使用される。本報告書ではメッセージ拡大を適用せずに処理を行うステップを「メッセージ拡大非適用ステップ」、メッセージ拡大を適用したメッセージを使用するステップを「メッセージ拡大適用ステップ」と呼ぶこととする。図 1 に記載されているパラメータ x, n の各ハッシュ関数における実際の値を表 1 に記載する。

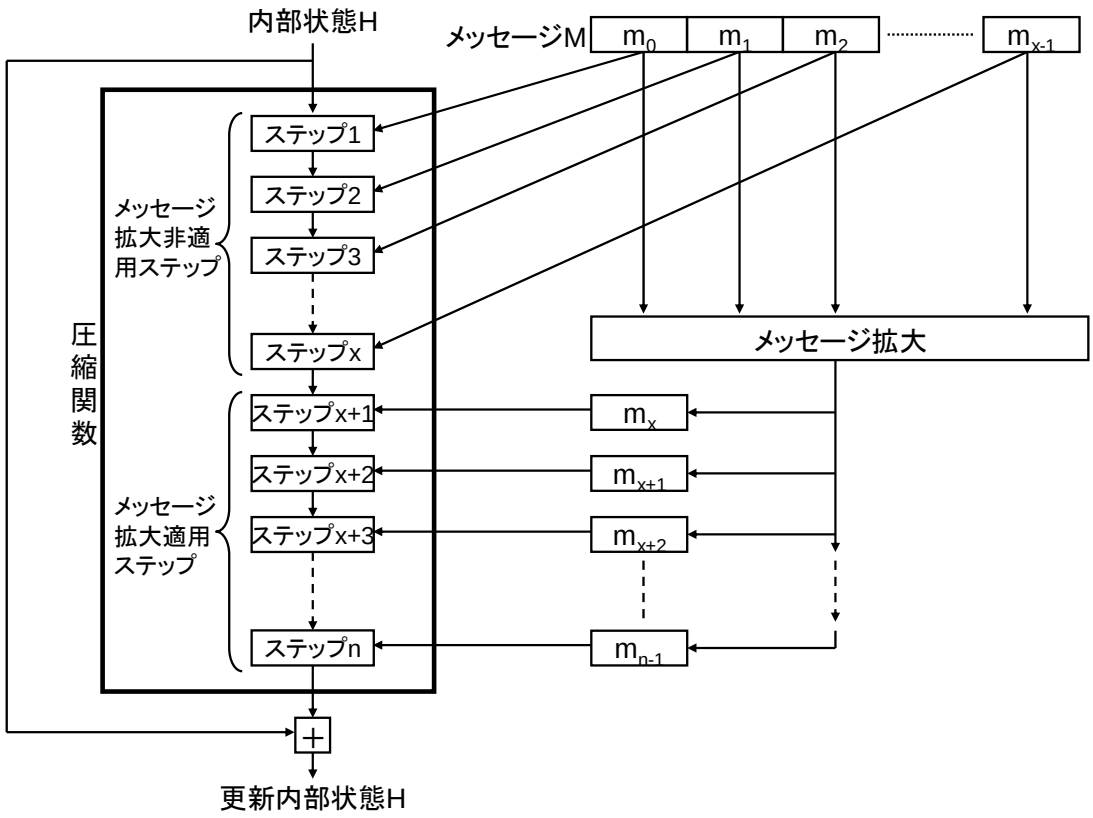

図 1. 一般的なハッシュ関数の構造

表 1. 各ハッシュ関数におけるパラメータ

ハッシュ関数	x	n
MD4	16	48
MD5	16	64
SHA-0	16	80
SHA-1	16	80
SHA-224	16	64
SHA-256	16	64
SHA-384	32	80
SHA-512	32	80

差分攻撃

「差分攻撃」は、共通鍵ブロック暗号に対する攻撃法として 1989 年に Biham らによって提案されたものであり、現在まで非常に多くの適用例が示されている。
差分攻撃とは、暗号アルゴリズム E にある固定の差分値を持つ二つの入力データを与えた時、各々の出力の差分値が高い確率である固定の値となる場合があるという性質を利用した攻撃法である。この確率を「差分確率」と呼ぶ。

差分攻撃では、暗号アルゴリズム E に対し、その入力 X の差分 ΔX、および出力 Y の差分 ΔY に対して、次の性質が成立する差分確率を p とした場合、差分攻撃に必要とされる計算量は、およそ 1/p であることが知られている。よって、差分確率が大きい程攻撃に必要な計算量が少なくてすみ、効率的な攻撃法となる。

$$E(X + \Delta X) = Y + \Delta Y$$

繰り返し型の内部構造を持つ暗号アルゴリズム E に対して差分攻撃を行なう場合(各一単位 E_i を「ステップ」と呼ぶ)、各ステップ毎に定められた入出力差分 $(\Delta X_i \rightarrow \Delta X_{i+1})$を満たしながら、最終出力差分に至る手法が一般的に用いられている。このような入出力差分の列を「差分パス」とよぶ。差分パスは局所的に有効な差分を効率的に繋げることによって、全体の差分確率が高いものを効率的に見付けるのに有効である。

$E(x) = E_n(...E_3(E_2(E_1(x))))$ である時、

$$\Delta X = \Delta X_0 \xrightarrow{E_1} \Delta X_1 \xrightarrow{E_2} \Delta X_2 \xrightarrow{E_3} \ldots \xrightarrow{E_n} \Delta X_n = \Delta Y$$

ハッシュ関数に対する攻撃法として現在までに知られているもののほとんどは、差分攻撃をベースとしたものである。差分攻撃の入力差分としてメッセージ差分、出力としてハッシュ値の差分を指定する。特に、コリジョン探索攻撃の場合、出力差分 = 0 となる差分パスで、成立確率が可能な限り高いものを見付けることが求められる。

マルチブロックコリジョン探索

SHA-1 に代表される Merkle-Damgard 型ハッシュ関数は、図 1 に示すように、メッセージを固定ビット長の「ブロック」に区切り、各ブロックの各先頭から順に「圧縮関数」のメッセージ入力に代入し、その出力を次の圧縮関数の入力とするアルゴリズムである。

1 ブロック分のメッセージで成り立つコリジョンを「ワンブロックコリジョン」、複数ブロック分のメッセージで成り立つコリジョンを「マルチブロックコリジョン」と呼び区別する。マルチブロックコリジョン探索とは、マルチブロックコリジョンが成り立つような複数ブロックからなるメッセージを探索するものである。

ローカルコリジョン

「ローカルコリジョン」とは、ハッシュ関数内部の圧縮関数に代入される各ステップのメッセージの差分値が各ステップ毎に独立した値に設定可能であると仮定した時、局所的にコリジョンを発生させる差分パスならびにその差分を最小ステップ数かつ高い確率で成り立たさせる為の条件式の集合(コンディション)を指す。これは入力した差分を最も短いステップ数で打ち消す方式であるとも言える。

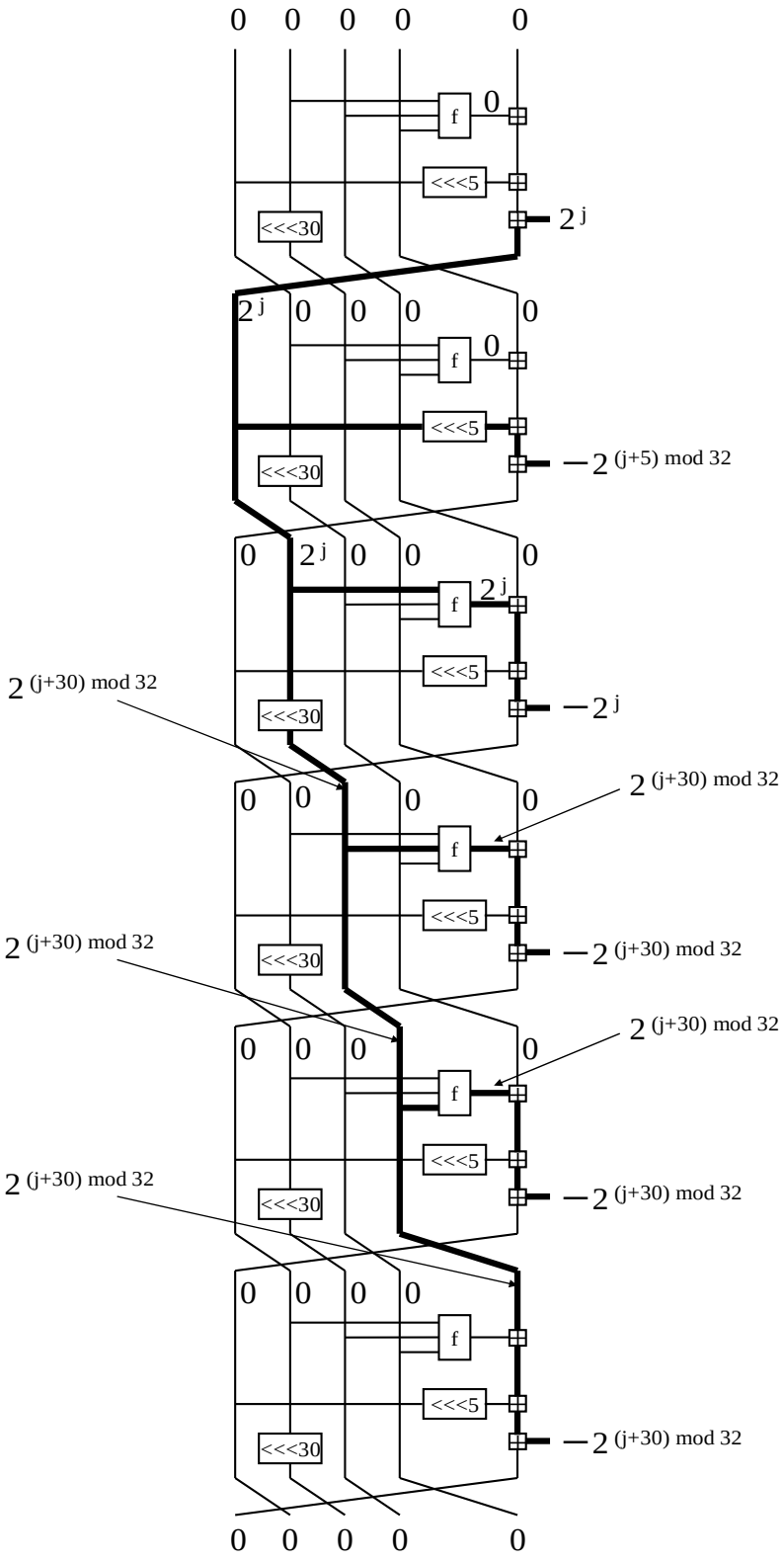

図 2. SHA-1 におけるローカルコリジョンの例(定数加算は省略)

ディスターバンスベクトル

ローカルコリジョンは、ステップ毎のメッセージ差分の独立性を仮定して構成されているが、実際のハッシュ関数では、各ステップ毎のメッセージには従属関係があり、単一のローカルコリジョンでハッシュ関数全体のコリジョンを生成することは出来ない。「ディスターバンスベクトル」とは、ローカルコリジョンの組合せによりハッシュ関数全体の差分パスを構成する際に用いられる技術であり、一つのローカルコリジョンをある 1 ビットの値で代表させることで、ローカルコリジョンを用いた差分パスを、メッセージ拡大を考慮した形でハッシュ関数全体に拡張して表現するのに適している。パータベーションマスクとも呼ばれる。

コンディション

ハッシュ関数の各ステップの入出力データをチェイニングバリアブル、入力メッセージをメッセージと呼ぶ。ハッシュ関数のコリジョン探索において構成した差分パスについて、その差分パスに関係する、チェイニングバリアブル、あるいはメッセージに付随するビット単位の条件式が全て成立すれば、高い確率で求める差分パスとなることが知られている。この条件式を「コンディション」と呼ぶ。メッセージに対して付与されるコンディションを「メッセージコンディション」、各ステップの入出力データに対して付与されるコンディションを「チェイニングバリアブルコンディション」(又はサフィシェントコンディション)と言う。本文において、単にコンディションと言う場合は、チェイニングバリアブルコンディションを指すこととする。チェイニングバリアブルコンディションの個数がコリジョン探索計算量に直接的に関係している。

メッセージモディフィケーション

メッセージモディフィケーションは、コリジョン探索において、差分パスを成立させるために必要なコンディションが成立していなかった場合に、一部のメッセージを修正することで、コンディションを満たすようにする手法である。メッセージ拡大非適用ステップ(非線形パート)に付随するコンディションに対して適用する「ベーシックモディフィケーション」とメッセージ拡大適用ステップ(線形パート)に付随するコンディションに対して適用する「アドバンストメッセージモディフィケーション」がある。

2. 算術演算をベースとするハッシュ関数の脆弱性に関する調査

本章では、算術加算をベースとした MD5, SHA-1 等のハッシュ関数について、ここ数年で指摘された差分解読法の実現可能性と、それが実現した際の暗号プロトコルやアプリケーションへの影響について調査する。

2.1. ハッシュ関数の危殆化による暗号プロトコルやアプリケーションへの影響

2.1.1. Fingerprint

ハッシュ関数の最も基本的な応用として fingerprint が挙げられる。多くの通信プロトコルやアプリケーションでは、扱うドキュメントや画像、音声等あらゆる電子データに対して、各々異なる識別子(固定ビット長)を付与し、データを区別する必要がある。この電子データ固有の識別子を電子データに対する fingerprint と呼ぶ。

もし、あらゆる電子データに対する fingerprint が各々固有の値であり、また電子データから誰でも計算可能であれば、そのデータが唯一である証拠として用いることができる。また、電子データが何らかの手段によりその一部でも改竄された場合は、fingerprint の値が異なることから、fingerprint は改竄検知にも利用される。

fingerprint は SSL などでの証明書の正当性の保証, PGP の公開鍵の本人性保証等で利用されており、改竄検知やなりすまし防止に役立てられている。fingerprint で利用されるハッシュ関数としては、MD5 や SHA-1 が多く用いられている。

その一方、脆弱なハッシュ関数を fingerprint の計算に用いた場合、さまざまな偽造例が示されている。

文献[1][2]では、PostScript 言語や TIFF フォーマット等 特定フォーマット文書の性質を利用し、視覚表示が全く異なる二つドキュメントで、MD5 による fingerprint を等しくする手法が示されている。図 3 は、PostScript で記述される文書の偽造方法について図示したものである。偽造の手順は次の通りである。

偽造の手順

1. PostScript 制御文字列に続く二種類のメッセージ R1, R2 について、そのハッシュ値が一致するデータを生成する。
2. 二つの意味のある PostScript ファイル file1,file2 を生成する。
3. 二つの文書ファイル A,B を図に示した通りに作成する。

この方法で作成した PostScript ファイルのハッシュ値は 1 の性質によって R1 あるいは R2 までのデータに対し一致し、続くファイルデータは共通であるから、結果として、ファ

イル A,B の fingerprint は一致する。

さらに文献[3]では、MD5 に対するターゲットコリジョンと呼ばれる手法を応用することにより、更に偽造可能なデータフォーマットを Microsoft WORD や Adobe PDF へ拡張することができると主張している。一例として異なる 12 個の PDF ファイルに対して、同一の MD5 fingerprint を持たせたものが記載されている。ここで示されている例は、2008 年に選出される予定のアメリカ大統領選挙の結果を 2007 年の時点で予測するというやや刺激的な内容であるが、その種明かしも次の通り記載されている。めぼしい候補者を各々記載した PDF ファイルに対して、PDF ファイル形式が持つ冗長性を利用し、全ての PDF ファイルが同じ MD5 出力値となるようパディングを行なうというものであり、ハッシュ関数 MD5 の脆弱性を利用したものである。

このように、脆弱名ハッシュ関数を用いた場合、もはや fingerprint によるデータの唯一性保証はなされないことから、安全な電子署名としては利用できないことが分かる。

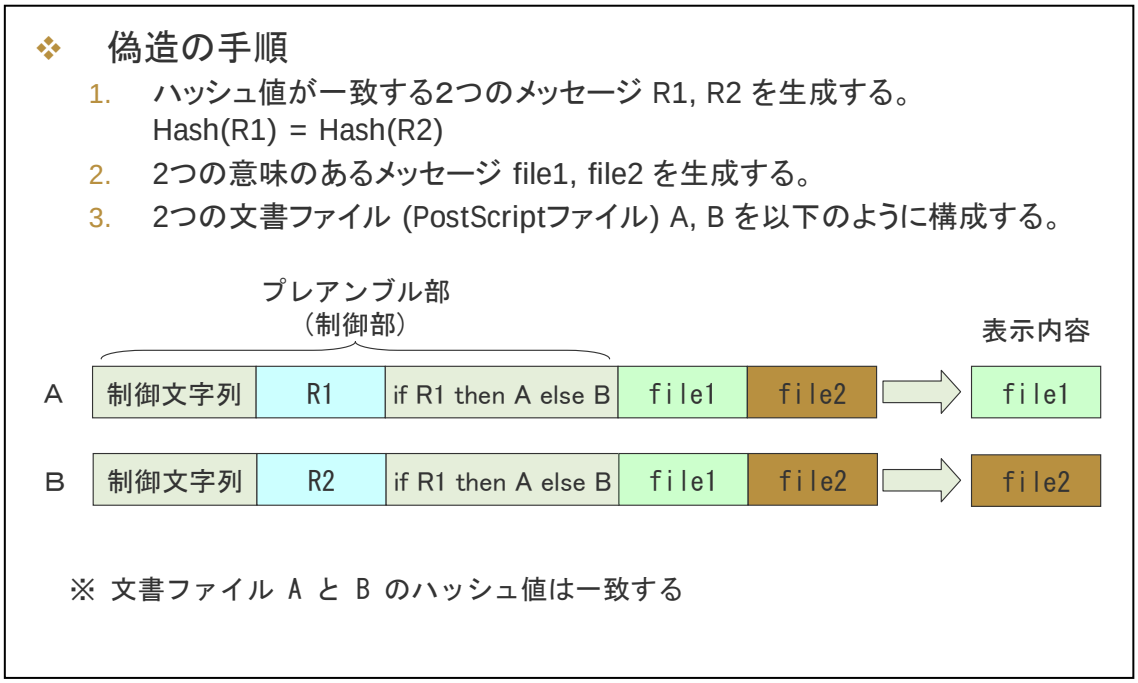

図 3. PostScript ファイルの MD5 fingerprint の偽造方法[2]

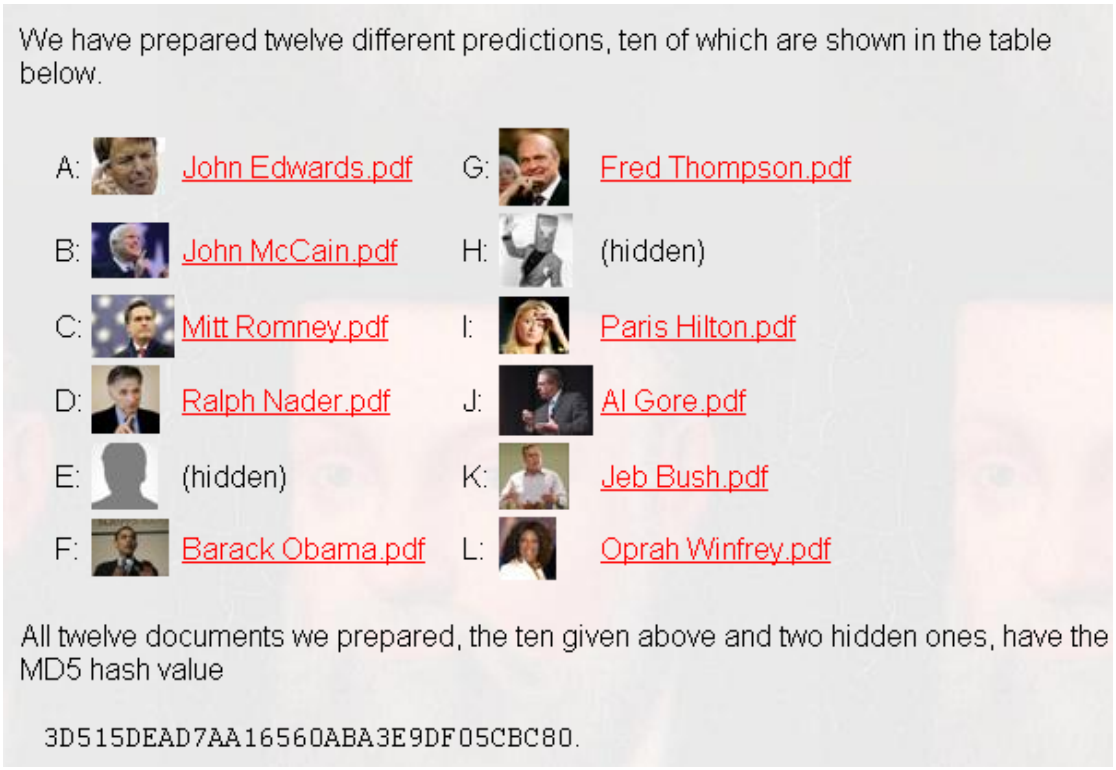

図 4. PDF ファイルに対する MD5 fingerprint の偽造データ一覧[3]

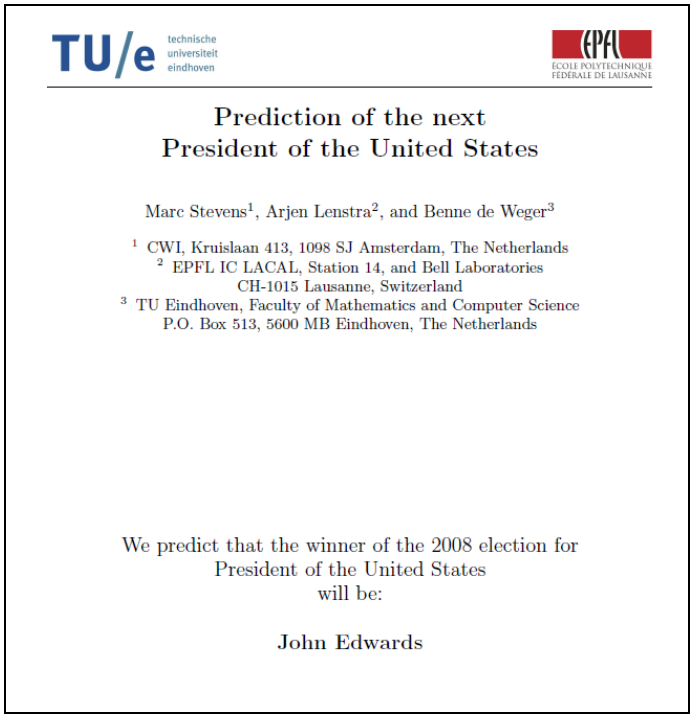

図 5. 等しい MD5 fingerprint を持つ PDF 文書の一つ(JohnEdwards.pdf)[3]

2.1.2. APOP

APOP[4]は、電子メールクライアントとサーバ間のパスワード暗号化プロトコルであり、処理の内部でハッシュ関数 MD5 を用いることが規定されている。APOP は，電子メールのクライアントとサーバの間で、チャレンジ・レスポンス方式に基づいた通信を行い、クライアントを認証するプロトコルであり、パスワードが通信路上を直接流れないようにしたものである。APOP プロトコルは次の通りである。

APOP プロトコル

1. サーバからクライアントに乱数(challenge code)を送付
2. クライアントは challenge code ならびにパスワードを連結
3. クライアントは連結した数列をハッシュ関数 MD5 を用いてハッシュ値を計算
4. クライアントは計算されたハッシュ値をサーバに送付
5. サーバは，クライアントと同様に、内部で保持するクライアントのパスワードからハッシュ値を計算
6. サーバは 4 と 5 で得られた数値が一致すれば、クライアントが正しいパスワードを持つと判断

文献[6][7]において、現実的な計算量でパスワードが漏洩することが示されている。本攻撃は、中間者攻撃が可能であること、すなわち攻撃者によって偽造されたメールサーバに対して、クライアントが APOP 認証を複数回試みることを仮定する。具体的な攻撃手法は次の通りである。なお、”||”はデータの結合処理を表す。

攻撃手法

1. 偽造サーバは、クライアントのパスワードの一文字目を、例えば「P」と予測し、63 バイトの数値 C1,C2 で、両者の数値の最後に`P'を付加した数列のハッシュ値が等しくなる値を求めておく。
   MD5(C1 || `P') = MD5(C2 || `P')
2. 偽造サーバは、クライアントからの APOP 認証要求に対し C1 を送り、クライアントから送られた MD5 ハッシュ値を保持し、再度の APOP 認証要求に対し C2 を送りクライアントから送られた MD5 ハッシュ値を保持する。
3. 偽造サーバは、この二つのハッシュ値が等しい場合にパスワードの最初の文字の予測が正しかったとみなし、そうでなかった場合は推定する文字を変えて、1 から繰り返す。
4. 偽造サーバは、二文字目について例えば`a'と予測し、62 バイト数値 C3,C4 で、両者の数値の最後に既に得られた一文字目とつなげた`Pa'を付加した数列のハッシュ値が等

しくなる値を求めておく。

5. 偽造サーバは、2,3 と同様の手順で、二文字目が正しく推定されているかを確認する。
6. 偽造サーバは、4,5 と同様に 3 文字目も推定する。

1 文字が英大小文字と数字で出来ているとすれば，偽サーバに約 60 回アクセスさせることで，1 文字分のパスワードが解析できることになる。文献[7]による攻撃法では、偽造サーバが得られるパスワードは最初の 3 文字までであるが、その後発表された文献[6]では、Wang らによる MD5 コリジョン探索に加え、Boer らによる MD5 擬似コリジョン探索(等しいメッセージかつ異なる IV から同一の MD5 の出力を得る方法)を用いることで、現実的な時間内で 31 文字以下のパスワードが暴かれることが示されている。また、文献[8]では、challenge code とパスワードの並びを入れ替えた場合[5]でも攻撃が可能であることが示されている。

2.1.3. HMAC, NMAC

HMAC および NMAC は 1996 年に Bellare, Canetti および Krawczyk らによって提案されたハッシュ関数をベースとしたメッセージ認証コード (MAC) である[9]。HMAC は TLS, SSH, IPsec 等に実装され、広く使われている方法である[10][11][12]。HMAC および NMAC は、ある条件の元で安全であることが証明されているが[13]、その一方で、MD4 を用いた HMAC-MD4 ならびに NMAC-MD4、MD5 を用いた NMAC-MD5 についてディスティングイッシュ攻撃[14]、更にはキーリカバリ攻撃[15][16][17][18]が可能であることが示されている。

HMAC および NMAC の演算方法は次の通りである。一般的なハッシュ関数では、初期値 IV, メッセージ m を用いてハッシュ値を生成する。この処理を H(m)と表記する。IV は MD4, MD5 など、ハッシュ関数ごとに定められている固定値である。このような H(m)に対し、変数 h を初期値 IV の代わりに使用するようなハッシュ処理を F(h, m)とすると、NMAC, HMAC は以下のように定義される。処理のイメージを図 7、図 6 に示す。なお、"||"はデータの結合処理を表す。

HMAC の演算

HMAC(k,text) = H((k XOR opad) || H((k XOR ipad) || text))

ここで、ipad = バイト値 0x36 を B(ハッシュ関数のブロック長)回繰り返した文字列

opad = バイト値 0x5C を B(ハッシュ関数のブロック長)回繰り返した文字列

text = メッセージ

k = 秘密鍵

NMAC の演算

NMAC(k1,k2)(text) = F(k1, F(k2, text))

ここで、{k1, k2} = 秘密鍵

text = メッセージ

NMAC と HMAC の関係

HMAC (k,m) = NMAC(k1,k2)(m)

ここで、k1 = H(k XOR opad), k2 = H(k XOR ipad)

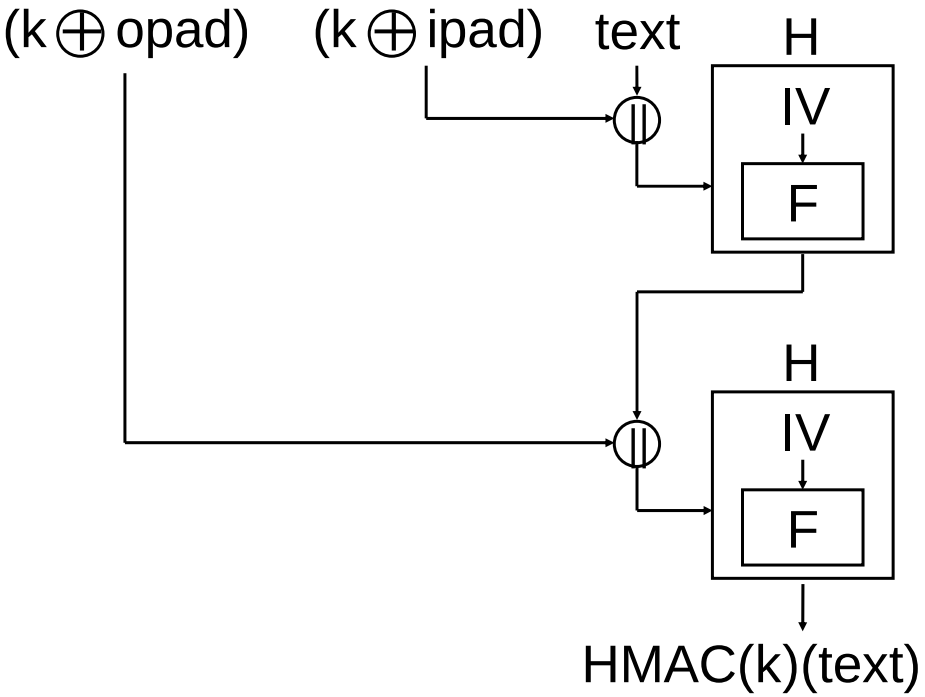

図 6. HMAC の処理

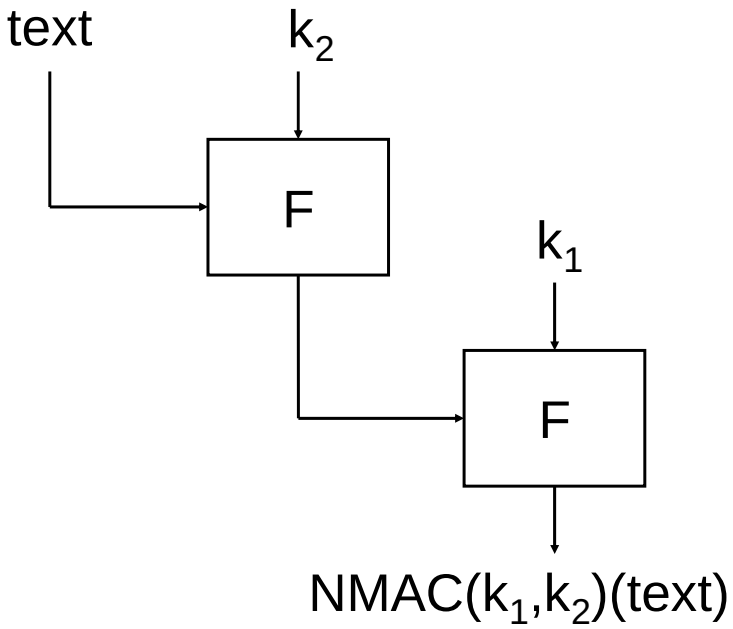

図 7. NMAC の処理

表 2. HMAC, NMAC に対する攻撃計算量

攻撃対象	攻撃手法	計算量	必要メモリ量	文献
HMAC-MD4,NMAC-MD4	ディスティングイッシュ攻撃	2^{58}		[14]
HMAC-MD4,NMAC-MD4	部分鍵回復攻撃	2^{63}	2^{40}	[15]
HMAC-MD4,NMAC-MD4	ユーザ鍵回復攻撃	2^{88}	2^{95}	[16]
NMAC-MD5 関連鍵	ディスティングイッシュ攻撃	2^{47}		[14]
NMAC-MD5 関連鍵	部分鍵回復攻撃	2^{47}	2^{45}	[15]
NMAC-MD5 関連鍵	ユーザ鍵回復攻撃	2^{51}	2^{100}	[16][17]
HMAC-SHA0,NMAC-SHA0	ディスティングイッシュ攻撃	2^{84}		[14]
HMAC-SHA0,NMAC-SHA0	部分鍵回復攻撃	2^{84}		[15]
HMAC-SHA1,NMAC-SHA1 (reduced 34 rounds)	ディスティングイッシュ攻撃	2^{34}		[14]
HMAC-SHA1,NMAC-SHA1 (reduced 34 rounds)	部分鍵回復攻撃	2^{34}		[15]

2.1.4. X.509

X.509 は ITU (International Telecommunication Union)や RFC 等で標準化された公開鍵証明書フォーマット[21][22]であり、S/MIME や SSL/TLS、などの多くのセキュリティプロトコルが X.509 をベースにしている。1988 年にバージョン 1、1993 年にバージョン 2 が公開されているが、現在は 1996 年に公開されたバージョン 3 が主に使われている。

文献[23][24][25][26][27]では、この X.509 証明書において MD5 等脆弱なハッシュ関数が利用されている場合、所有者と公開鍵が異なる以外、全て同じ X.509 証明書の組の作成が現実的な時間内で可能であることが示されている。

本攻撃シナリオは以下の通りである。

攻撃シナリオ

(1) X.509 証明書の MD5 の「ターゲットコリジョン」と呼ばれる攻撃を実施する。
(2) コリジョンデータを公開鍵情報の部分に埋め込む。
(3) 異なる二つの X.509 証明書と X.509 証明書を生成し一方を正規に登録する。
(4) 攻撃者は、偽造 X.509 証明書を利用して電子商行為を行う。

ターゲットコリジョンとは、サイズは等しいが内容が異なる任意のデータの組（ターゲット）に対して、それぞれの末尾に攻撃者が適切なデータを繋げることにより、ハッシュ値を衝突（コリジョン）させる攻撃技術である。単にハッシュ関数のコリジョンデータを求めるよりも多くの計算量が必要であると考えられる。

この偽造された X.509 証明書の署名は正当な X.509 証明書の署名と等しいため、これら 2 種類の証明書が悪用されれば、X.509 証明書の否認不可性が破れることを意味する。MD5 の場合 X.509 証明書の偽造にかかる計算量は 2^{50} 回程度であると述べられており[25]、コリジョン探索計算量が少なければ、現実的な計算量での偽造が可能となる。また、本攻撃法は、ターゲットコリジョンが成立すれば、SHA-1 の場合であっても証明書偽造が可能であると述べられている。

本攻撃法では、正当な証明書と偽造証明書の所有者は異なる攻撃者は正当な証明書と偽造証明書を同時に生成しなければならないが、特に SHA-1 を用いた X.509 証明書プロトコルはさまざまな所で利用されており、情報社会インフラの核である電子証明書の偽造が、脆弱なハッシュ関数を用いることで可能となる場合があるという点は、その社会的影響は甚大であり、多いに憂慮すべきと考えられる。そこである仮説の元、SHA-1 のコリジョンが発見されてから、SHA-1 を用いた証明書の偽造が成功するまでの期間に付いて考察する。

MD5 の現時点で知られているコリジョン探索計算量は 2^{29} であり[61]、MD5 を用いた X.509 証明書偽造計算量とは、2^{21} 程度の開きがある。この差は主として、下記による計算量の増

加が起因している。
・ターゲットコリジョン探索に必要な計算量の増加
・RSA 公開鍵として有効な値（二つの素数の積）が導かれるまでの探索繰り返しによる計算量の増加
SHA-1 のコリジョン探索計算量と SHA-1 を用いた X.509 証明書の偽造計算量の比較に関する知見は現時点で得られていないが、例えば仮に MD5 の場合と同様、SHA-1 の X.509 証明書偽造に必要な計算量も SHA-1 コリジョン探索計算量に対して 2^{20} 程度の開きがあるとの仮説の元、コリジョン探索実行時と同程度の費用ならびに同程度の期間の計算資源を用いると仮定した場合、SHA-1 を用いた X.509 証明書偽造が成功するまでに、ムーアの法則（1.5 年で 2 倍）を考慮して 20×1.5 年=約 30 年のタイムラグがあると推定される。以上の考察より、SHA-1 のコリジョンが発見されたとしても、SHA-1 に対する劇的なコリジョン探索計算量削減手法が提案されない限り、SHA-1 を用いた公開鍵証明書の偽造が直ちに可能となるとは考えにくい。
本評価は、あくまでもいくつかのの仮説に基づいたものであり、より厳密に行うべきであるのは明らかであるが、そのためには SHA-1 コリジョン探索、ならびに SHA-1 を用いた X.509 証明書偽造の各々の計算量に対し、より詳細な評価が急務である。

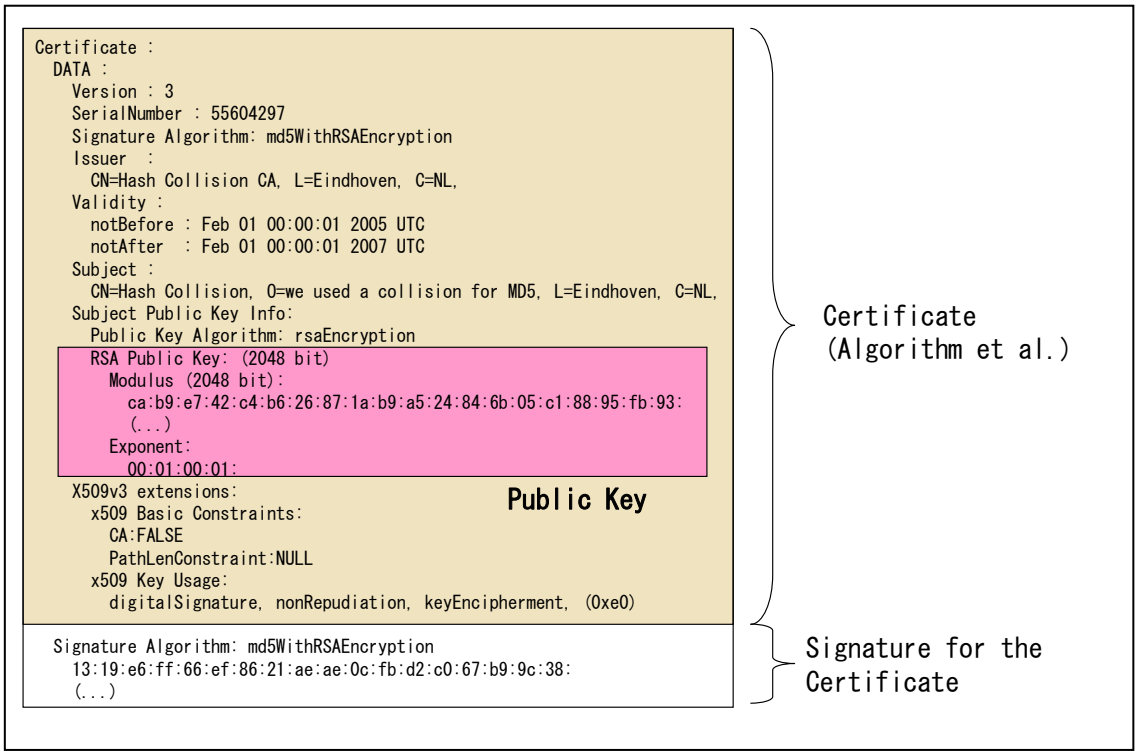

図 8. X.509 署名証明書

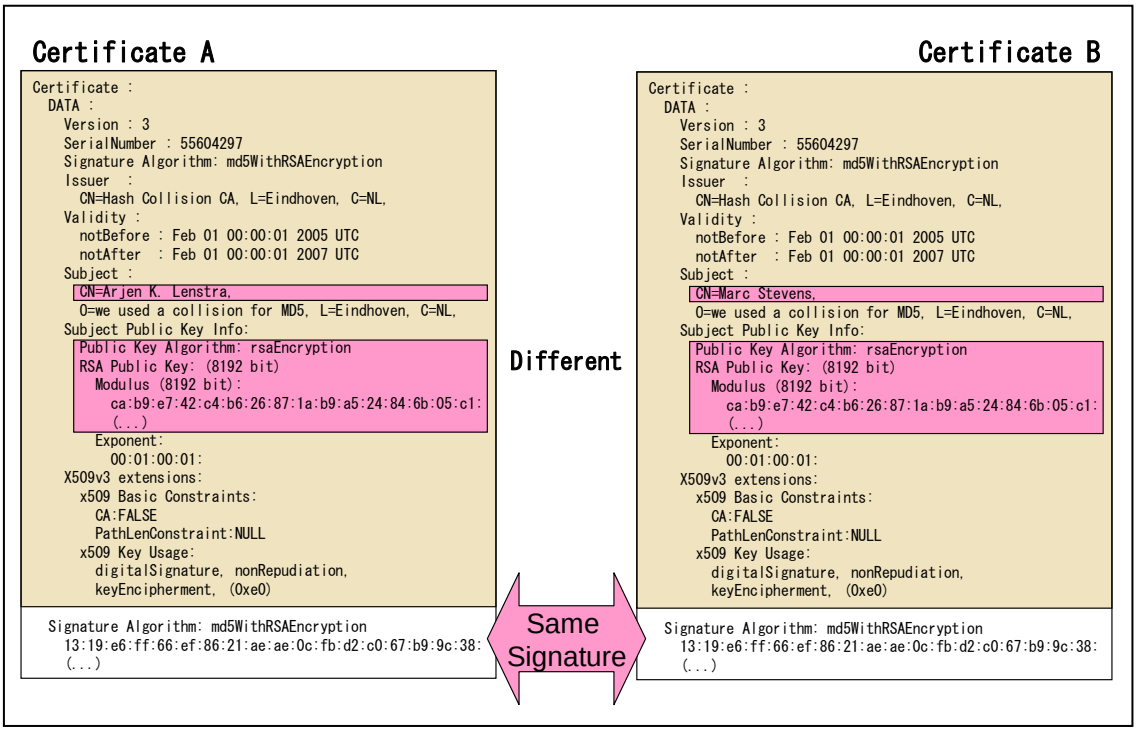

図 9. X509 署名の偽造例

2.1.5. IPSEC

IPsec(IP Security)は、IETF(Internet Engineering Task)において, IP(Internet Protocol)レベルの暗号化機能として標準化されたものであり、認証や暗号のプロトコル，鍵交換のプロトコル, ヘッダー構造など, 複数のプロトコルを総称するものである 0[38][39][40][41]。

IPsec(VPN)通信を行うには，通信相手（ピア）との間で Security Association(SA)と呼ばれる論理的なコネクションを確立する。SA は VPN 通信を行うトラフィック毎に確立され，トラフィック情報(selector)と，暗号アルゴリズム，認証アルゴリズム等、トラフィックに適用するセキュリティ情報を含む。SA を確立した後，ルータは SA の情報に基づいて VPN 通信処理を行う。自動鍵管理プロトコルを使用した場合，対象パケットデータ受信を契機に自動的にピアとネゴシエーションを行って鍵を交換し，SA を確立する。IPsec SA 上での通信は IPsec SA 確立時に交換した暗号アルゴリズムと鍵を用い, 暗号化ならびに復号を行なう。

IPsec ならびに IKE では暗号化機能を実現するための部品として暗号学的ハッシュ関数を利用しているが、ここ数年の間に相次いで発表されたハッシュ関数の脆弱性発見に関する対応について、2007 年 5 月に、IKE（Internet Key Exchange）および IPsec におけるハッシュアルゴリズムの使用に関する文書が RFC 4894 として承認された[43]。

RFC 4894 は IKEv1 と IKEv2、IPsec プロトコルがどのようにハッシュ機能を使うかについて述べている。また、MD5 および SHA-1 アルゴリズムの劣化した衝突耐性について、こうしたプロトコルの脆弱性がどのような水準にあるのかについても説明している。

IKEv1, IKEv2, IPsec におけるハッシュ関数の脆弱性の影響について指摘されている事項をまとめたものを表 3 に示す。

表 3. ハッシュ関数の脆弱性に関する IPsec のセキュリティへの影響[43]

プロトコル	ハッシュ関数の使用用途	影響
IKEv1	疑似乱数生成	影響無し
	疑似ランダム置換	影響無し
	X.509 電子署名	MD5 のターゲットコリジョン攻撃が適用可能
	公開鍵確認	影響無し
	NAT(IP アドレス隠蔽)	影響無し
	PKIX 証明書	MD5 のターゲットコリジョン攻撃が適用可能
IKEv2	疑似乱数生成	影響無し
	HMAC	MD5,SHA-1 で鍵回復攻撃が適用可能
	X.509 電子署名	MD5 のターゲットコリジョン攻撃が適用可能
	公開鍵認証	影響受けにくい
	NAT(IP アドレス隠蔽)	影響無し
	PKIX 証明書	MD5 のターゲットコリジョン攻撃が適用可能
IPsec	HMAC	MD5,SHA-1 で鍵回復攻撃が適用可能

2.1.6. PKCS

PKCS(Public Key Cryptography Standards) は RSA Laboratories によって策定されている RSA 公開鍵暗号ベースの暗号化および認証プロトコルである[28]。PKCS は #1 から #15 まであるが、2008 年 2 月現在のカレントバージョンならびにこれらにおけるハッシュ関数の利用状況は表 4 の通りである。

表 4. PKCS におけるハッシュ関数の利用状況

PKCS クラス	バージョン	記載
PKCS #1	v2.1	AppendixB において、利用可能なハッシュ関数リストが掲載されている。RSAES-OAEP, EMSA-PSS, RSASSA-PKCS, RSASSA-PSS では SHA-1, SHA-256/384/512 が推奨されているが、MD2, MD5 も互換性を理由に記載されている。
PKCS #3	v1.4	ハッシュ関数に関係する記載はない
PKCS #5	v2.0	PBKDF1 では MD2, MD5, SHA-1、HMAC では SHA-1,SHA-256/384/512 が記載されている。
PKCS #6	v1.5	ハッシュ関数に関係する記載はない
PKCS #7	v1.5	ハッシュ関数に関係する記載はない
PKCS #8	v1.2	ハッシュ関数に関係する記載はない
PKCS #9	v2.0	ハッシュ関数に関係する記載はない
PKCS #10	v1.7	ハッシュ関数に関係する記載はない
PKCS #11	v2.2	DSA, ECDSA では SHA-1, SHA-2 が記載されている。#1v1.5 の RSA-PKCS で MD2, MD5, SHA-1/256/384/512, RIPEMD-128/160 の記載が引用されている。HMAC で MD2, MD5, SHA-1/256/384/512, RIPEMD-128/160 が記載されている。
PKCS #12	v1.0	#5 の HMAC で SHA-1 の記載が引用されている。
PKCS #13	v1.0	ドキュメント未完のまま
PKCS #15	v1.1	Credential Identifiers で SHA-1 が記載されている。

2.1.7. SSL/TLS

SSL(Secure Socket Layer)[31]、TLS(Transport Layer Security)[32][33]の RSA 暗号を用いた公開鍵暗号規約は PKCS#11 v2.0 に記載されているが、その公開鍵に対する証明書の形式としては、X.509 公開鍵証明書を用いることが規定されている。

この X.509 証明書については、前述の MD5 に対するターゲットコリジョンを用いた偽造により、改竄される危険性が指摘されている。従って SSL/TLS の公開鍵証明書に MD5 等脆弱なハッシュ関数を用いると、サーバの偽造が可能となる場合がある。ただし、本攻撃法による SSL/TLS サーバの改竄が成功するには、公開鍵証明書発行時点で、二種類の X.509 公開鍵証明書を準備する必要がある。

2.1.8. Timestamp

タイムスタンプはある時点でのドキュメントの存在を保証する暗号スキームである。タイムスタンププロトコルは、PKCS#1v1.5 をベースとした RSA 署名スキームの応用であり、RFC3161 で規定されている[35]。ユーザがドキュメント m に対するタイムスタンプを要求する場合、以下のように処理を行う。

タイムスタンプの処理

(1) ユーザはドキュメントのハッシュ値 H(m)と 64 ビットの乱数をタイムスタンプサーバに送る。

(2) タイムスタンプサーバは、以下の処理を行う。

(i) TSTInfo と呼ばれる情報を生成する。TSTInfo は、ユーザから送られてきたハッシュ値 H(m)(messageImprint)、乱数 (nonce)、およびタイムサーバの現在時刻(genTime)が含まれる。

(ii) タイムスタンプサーバ自身の情報と TSTInfo のハッシュ値(messageDigest)で signedAttrs と呼ばれる情報を生成する。

(iii) タイムスタンプサーバ自身の(署名用の)秘密鍵を用いて signedAttrs に対する署名 signature を生成する。

(iv) TSTInfo、signedAttrs、signature を結合したものをタイムスタンプトークン(TST)として返送する。

(3) 検証者は、タイムスタンプトークンを検証することにより、genTime に記述された時間にドキュメントが存在したことを検証できる。

タイムスタンププロトコルの内部で X.509 規格に基づく RSA 公開鍵証明書を用いているため、コリジョンが求められるような脆弱なハッシュ関数を用いた場合には、X.509 に対する攻撃法と同様の方法で証明書の偽造が可能となる可能性があり、MD5 や SHA-1 等、脆弱なハッシュ関数を用いている場合には、ドキュメントの時刻情報を改竄されるおそれがある。ただし、日本国内においては、タイムスタンププロトコルに基づく認証業務を行う際、SHA 系ハッシュ関数を使う場合には SHA-256 以上のビット長を持つハッシュ関数を用いることが規定されており[1]、MD5, SHA-1 は含まれていない。

なお、タイムスタンププロトコルについては、PKCS#1v1.5 の実装の脆弱性を利用した攻撃法が知られている[36]。ただし本攻撃にハッシュ関数の脆弱性は利用されていない。

[1]日本データ通信協会タイムビジネス協議会、“(改定) タイムビジネス認定基準、時刻認証業務デジタル署名を使用する方式”、第 3 回制度諮問委員会, 2006. 09. 01,
http://www.dekyo.or.jp/tb/data/D-Rcriteria070126.pdf

3. 算術演算をベースとするハッシュ関数の差分攻撃に関する調査

本調査では、算術演算をベースとしたハッシュ関数の安全性について明らかにするため、差分攻撃法をベースとしたローカルコリジョン探索法に関して調査を行うとともに、差分解読法のローカルコリジョン選択を現実的な時間内で終了するための課題を抽出する。

3.1. 差分解読法の計算量削減に関するローカルコリジョンの選択方法

ローカルコリジョンの概念は、文献[79]において、SHA-0 に対し差分攻撃法を効率的に適用する手段として提案されたものであり、続く SHA-0 への差分攻撃に関する文献[67][68][69][70][71]および、SHA-1 への差分攻撃に関する文献[75][76][77]、また文献[78]で SHA-256 に対してローカルコリジョンの適用がなされている。

ローカルコリジョンとは、ハッシュ関数内部の圧縮関数に代入されるメッセージ差分が各ステップ毎に独立した値としてよいと仮定した場合に最短ステップ数で成立するコリジョンのことである。これは入力した差分を最も短いステップ数で打ち消す方式であるとも言える。(図 2 参照。)

現在知られている SHA-0, SHA-1, SHA-256 に対する差分攻撃は全て、ローカルコリジョンの組み合わせをベースとした差分パスを元に構築されており、算術差分をベースとするこれらのハッシュ関数に対する差分攻撃を行なう上では、最も基本的な項目であると考えられる。

ローカルコリジョンを求める方法として、従来知られている手順について、おおまかにまとめると、下記の通りとなる。

ローカルコリジョン導出手法

(1) 任意のステップにメッセージ差分を与える。
(2) 各ステップの出力差分を出来る限り打ち消すようメッセージ差分を与える。
(3) ステップの出力差分が 0 になるまで繰り返す。

3.2. 差分解読法の計算量削減による最適なローカルコリジョン選択の要件

ローカルコリジョンは、ハッシュ関数に対する差分攻撃法の基本的な構成要素である。ハッシュ関数の各ステップ内部で用いられるステップ内部で用いられる非線形関数（f 関数）によって、ローカルコリジョンを成立させるために何らかの条件が必要であり、その条件の成立確率に関する検討が必要になる。各条件式は、ビット単位の線形関係式として表現されていることから、文献[79][59][75][76][77]による方法としては、各条件式の成立確率を各々 1/2 とし、関係式の個数 n に対して、全体の成立確率を 2^{-n} であると評価する方法が考えられている。なお、ローカルコリジョンのを成立させるための条件の例は文献[90]などに記載されている。

上記考察を考慮した場合、適切なローカルコリジョンを選択するためのハッシュ関数が満たすべき条件として次が考えられる。

(1) 同一構造を持つステップを繰返すハッシュ関数であること。（ステップ内部の非線形関数がステップ毎に異なる場合にも適用可能。）
(2) ステップ関数内部で使われる非線形関数について、ローカルコリジョンで使用する入力差分と出力差分の組合せが各々成立可能であり、そのための条件式、ならびに成立確率が得られること。

一方、文献[80]では、SHA-1 のローカルコリジョンの成立確率に関して、算術加算のキャリーの影響が詳細に検討された検討結果が示されており、SHA-1 のローカルコリジョンの成立確率が機械的な見込みよりも、若干大きくなることが指摘されている。この現象は、表 5 のように特に隣り合う 2 ビットから生成させた 2 個のローカルコリジョンを成立させる場合にその確率が顕著に変化するという性質として現れる。しかし、影響は軽微であるため、安全性評価の概算値を得る上では、必須ではないと考えられる。

(3) 算術キャリーの影響を考慮した際のローカルコリジョン成立確率評価。

表 5. SHA-1 にて XOR 関数を f 関数に持つステップのローカルコリジョン組合せ($-2^1 + 2^0$)成立確率[80]

確率評価法	成立確率
コンディション数による評価	2^{-4}
算術加算キャリーの影響を考慮した評価	$2^{-3.6781}$

表 6. SHA-1 の 23 ステップ以降のコンディション成立確率[79]

確率評価法	成立確率
コンディション数による評価	2^{-66}
算術加算キャリーの影響を考慮した評価	$2^{-64.5683}$

3.3. ローカルコリジョン選択を現実的な時間内で終了するための課題抽出

差分攻撃を行なう為のローカルコリジョンの選択はハッシュ関数毎に行なわなければならない。従来のハッシュ関数である SHA-0, SHA-1, SHA-2 では、ローカルコリジョンの求め方はほぼ同一であり、その表現も、f 関数の差による成立条件式の違いを除き、ハッシュ関数毎にほぼ唯一と言ってよい。また、SHA-1 等のローカルコリジョンについては、全差分パスを尽くすまでもなく、手計算で求めることができるため、ソフトウェアツールを作成し機械的に求めるより、手順法則に従って論理的に求めた方が効率的であると考えられる。

結論としては、与えられた評価対象のハッシュ関数アルゴリズムに対して、ローカルコリジョンが存在する場合は、それを現実的な時間内で求めることに大きな課題はないと考えられる。ただし、ローカルコリジョンが非常に多く存在する場合については、効率的なパターン分類が必要になるかもしれない。以上より課題として次が挙げられる。

(1) 機械的導出と手作業による導出のトレードオフ
(2) ローカルコリジョンが複数パターン存在する場合の扱い

4. 算術演算をベースとするハッシュ関数のディスターバンスベクトルに関する調査

算術演算をベースとするハッシュ関数に対する差分攻撃が現実的に可能となるためには、算出されたローカルコリジョンから「ディスターバンスベクトル」を構成する必要がある。本章では、このディスターバンスベクトルの構成法に関して調査を行うと共に。現実的な時間内でディスターバンスベクトルを構成するための課題を抽出する。

4.1. ディスターバンスベクトル構成手法の調査

ローカルコリジョンは、ステップ毎のメッセージ差分の独立性を仮定して構成されているが、実際のハッシュ関数では、各ステップ毎のメッセージには従属関係があり、単一のローカルコリジョンでハッシュ関数全体のコリジョンを生成することは出来ない。

より具体的には、メッセージ拡大非適用ステップで入力されるメッセージ差分は、メッセージ拡大適用ステップにおけるメッセージ差分に依存して決まるため、これらを考慮した上で、ハッシュ関数全体で成立する差分パスを構成する必要がある。

ローカルコリジョンの組合せによりハッシュ関数全体の差分パスを構成する一つの方法として、ディスターバンスベクトルを利用する方法がある。ディスターバンスベクトルは、文献[79]においてハッシュ関数 SHA-0 に対する差分攻撃に利用する為にローカルコリジョンとともに提案された概念である。(文献[79]ではパータベーションマスクと呼ばれている。) その後 文献[75][81][82]において、SHA-1 に対する差分攻撃に適用されている。

ディスターバンスベクトルとは、あるステップのメッセージ x の差分 1 ビット x_i から派生する一つのローカルコリジョン差分パスを、その 1 ビット x_i のみで代表することで、メッセージ拡大を考慮した差分パスを、ローカルコリジョンの組合せによりハッシュ関数全体に拡張して表現する手段である。

文献[79]では、SHA-0 のステップ r のメッセージ W^r に関するメッセージ拡大アルゴリズムが、数式 1 のように

(1) 再帰的に表現されていること

(2) ビット間の線形式であること

を利用し、各ステップ r のディスターバンスベクトル X^r についても、メッセージ拡大と同じアルゴリズムが適用可能であることを利用している。この性質は、SHA-1 のメッセージ拡大アルゴリズムでも同様である。

数式 1. SHA-0 のメッセージ拡大処理

$$W^r = W^{r-3} \text{ XOR } W^{r-8} \text{ XOR } W^{r-14} \text{ XOR } W^{r-16}, \ (r >= 16)$$

数式 2. SHA-1 のメッセージ拡大処理

$$W^r = (W^{r-3} \text{ XOR } W^{r-8} \text{ XOR } W^{r-14} \text{ XOR } W^{r-16}) <<< 1, \ (r >= 16)$$

4.2. ディスターバンスベクトル構成アルゴリズムの要件

本節では、ディスターバンスベクトルを構成するアルゴリズムに必要な要件について述べる。

文献[79]では、差分攻撃に有効なもののみを探索することで、探索空間を減らす試みがなされている。SHA-0 に対する差分攻撃に有効なディスターバンスベクトルが満たすべき条件として、

(i) $X^{(-5)} = 0, \ldots, X^{(-1)} = 0,$

(ii) $X^{(75)} = 0, \ldots, X^{(79)} = 0,$

であることを挙げており、条件 (i), (ii) を同時に満たすものは、全探索空間 $2^{16} = 65536$ のうちわずか 128 個であったことが述べられており、この中から改めて最適なものを見付けるという手順が示されている。

条件 (i) (ii) は、全ステップをローカルコリジョンの組合せで構成する為に必要な条件であるが、これらの条件については、文献[75]で導出された非線形差分パスの概念、およびマルチブロックコリジョンの概念を利用することによってそれぞれ条件は不必要であることが示されている。

1st ラウンド、3rd ラウンドの f 関数 (IF 関数 と MAJ 関数) については、入力差分ビットから出力差分ビットを得る確率が 1/2 となるため、差分ビットは出来る限りが少ない方がよい。

表 7. 各 f 関数の差分確率

入力差分	出力差分=Δe となる確率		
f 関数	IF	MAJ	XOR
(Δe, 0, 0)	1/2	1/2	1
(0, Δe, 0)	1/2	1/2	1
(0, 0, Δe)	1/2	1/2	1
(Δe, Δe, 0)	1/2	1/2	0
(Δe, 0, Δe)	1/2	1/2	0
(0, Δe, Δe)	1	1/2	0
(Δe, Δe, Δe)	1/2	1	1

さらに算術加算のキャリーの影響が出来る限り押えられるようにするため、メッセージ差分については、最上位ビット(31 ビット) 以外に現れるビット差分の個数が出来る限り少ないものを選択した方がよいことが述べられている。

文献[75]において、SHA-0 の場合は、メッセージ拡大の性質から、各メッセージ 32 ビットについて、各々独立に探索可能であることから、ディスターバンスベクトルの全探索空間は、2^{16} となるため、全探索が容易であるが、その一方、SHA-1 の場合は、全てのメッセ

ージ拡大非適用空間全体が探索範囲となるため、探索空間は $2^{32\times16} = 2^{512}$ となり、全空間の探索は実質的に不可能である。

文献[79]では、SHA-1 のディスターバンスベクトルを効率的に探索する手段として次の方法が提案されている。

(1) 探索空間の選択
(2) 攻撃計算量の概算評価

(1) について、算術差分のキャリーの影響を出来る限り押えるには、メッセージ拡大の性質から下位 2 ビットまでにディスターバンスベクトルがあるものが望ましい。この性質より、ディスターバンスベクトル全候補を探索する代わりに、連続する 16 ステップの下位 2 ビットを全探索し、その空間を全 80 ステップに当てはめることで、有効と思われるディスターバンスベクトルについては、ほぼ尽くしているとみなしている。
(2) について、ディスターバンスベクトルの全候補について、攻撃計算量を導出することは困難であったことから、次の手段で簡易的に評価している。

1. ハミング重みによる足きり
2. カウンティングルール(表 8) の適用

文献[81][82]では、上記これら 1, 2 の評価法に関し、抽出されたデータが必ずしも最適ではないことが示されており、1,2 の代わりに次の評価法が提案されている。

3. ディスターバンスベクトルに従って、ローカルコリジョンを展開し、各ステップの差分パス成立確率をより精密に算出

表 8. コンディション数のカウンティングルール[75]

step	disturb in bit 2	disturb in other bits	comments
19	0	1	For a_{21}
20	0	2	For a_{21}, a_{22}
21	1	3	Condition a_{20} is ``truncated''
22-36	2	4	
37	3	4	
38-40	4	4	
40-60	4	4	
61-76	2	4	
77	2	3	Conditions are ``truncated''
78	2	2	starting at step 77
79	(1)	(1)	Conditions for step 79,80 can be
80	(1)	(1)	ignored in analysis.

[スペシャルカウンティングルール]

- ある step のディスターバンスベクトルについて、ビット位置 0, 1 共に ”1” がある場合、コンディション数は 4 とカウントする(ビット位置 0 は最下位、1 は最下位から 1 ビット上位をあらわす)。
- ラウンド 3 については、F 関数(MAJ)の性質により、2 step 連続で同一ビット位置に “1” があるディスターバンスベクトルのコンディション数は(8 ではなく) 6 とカウントする。

4.3. 現実的な時間内でディスターバンスベクトルを構成するための課題抽出

差分攻撃法を効率的に実施する上で、最も有効なディスターバンスベクトルを現実的な時間内で求める上で課題となる点は次の通りである。

(1) 探索空間の選択法
(2) 攻撃計算量の算出法

(1)について、前節で述べたように SHA-0 の場合は、メッセージ拡大の性質から、ディスターバンスベクトルの全探索空間は、2^{16} となるため全探索が容易であるが、その一方で SHA-1 の場合は、探索空間は 2^{512} となり、全空間の探索を現実的な時間内で終了させることは不可能である。そのため[75]で提案されたような、差分攻撃に有効なもののみを効率的に探索することで探索空間を減らす試みが必須である。ただし、文献[81][82]で指摘されているように、探索すべき空間を減らしすぎた場合、攻撃する上で最適な要素を見逃す危険があるため、注意が必要である。

(2) について、探索空間に含まれるディスターバンスベクトル候補全てに関し、各候補を用いた際の攻撃計算量を精密に評価することが可能であればよいが、そうでない場合はあらかじめ簡易的な篩にかけておく必要がある。簡易的なチェックの方法としては次の方法が考えられる。

- ディスターバンスベクトルの全ハミング重みが一定以下のものを選択する方法
- 線形パート（メッセージ拡大適用ステップ）のみ成立確率を評価する方法

5. 算術演算をベースとするハッシュ関数の差分パスに関する調査

算術演算をベースとしたハッシュ関数に対する差分攻撃が現実的に可能となるためには、「差分パス」の構成が必須となる。本章では、この「差分パス」に関する観点から分析を行い、差分パス生成ツールを実現する上で解決すべき課題を抽出する。

5.1. 差分パス探索の技術的問題点

ローカルコリジョンならびにディスターバンスベクトルは、ハッシュ関数全体で成立する、攻撃に有効な差分パスを効率的に導出する為に考え出された概念であるが、与えられたディスターバンスベクトルならびにローカルコリジョンから差分パスを具体的に構成する際にもさまざまな問題点が生じる。本節ではその問題点について述べる。
まず、差分パスは次の 2 種類があることを念頭に置く必要がある。

[A] メッセージ拡大非適用ステップの差分パス（非線形パス）
[B] メッセージ拡大適用ステップの差分パス　（線形パス）

大まかに言えば、[B]はローカルコリジョンを適用して構成する差分パスであり、例えば SHA-0, SHA-1 の場合はステップ 17 以降の差分パスを指す。それに対し、[A]は、ローカルコリジョンを考慮せずに、与えられた入力差分、出力差分、ならびにメッセージ差分の間で矛盾が起きないように張られた差分パスであり、例えば SHA-0, SHA-1 の場合は ステップ 1 からステップ 16 までを指す。

差分パスを構成する際には、それと共に差分パスを成立させる為のビット単位の条件が必要である。この条件式をコンディションと呼ぶ。メッセージに対して付与されるコンディションをメッセージコンディション、各ステップの入出力データに対して付与されるコンディションをチェイニングバリアブルコンディション（又はサフィシェントコンディション)と言う。本文において、単にコンディションと言う場合は、チェイニングバリアブルコンディションを指すこととする。このチェイニングバリアブルコンディションの個数がコリジョン探索計算量に直接的に関係する。次節ではこのコンディションをより高い確率で成立させるための手法として、メッセージモディフィケーションと呼ばれる技術を適用する。[A]に関係するステップでは、次節のベーシックメッセージモディフィケーションを適用し、[B]に関係するステップではアドバンストメッセージモディフィケーションを適用する。

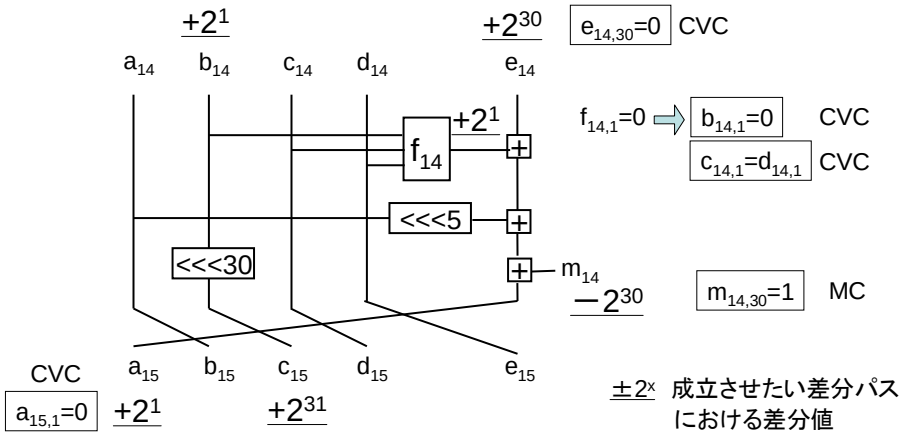

図 10. チェイニングバリアブルコンディションの例([95]より)

ハッシュ関数の差分パスを構成する上で、必要となる情報は以下の通りである。

(a) ハッシュ関数アルゴリズム
(b) ローカルコリジョン
(c) ディスターバンスベクトル
(d) メッセージ差分の符号の決め方

(a), (b), (c) に関する従来の検討結果については前節までに述べているため、ここでは (d) について述べる。ディスターバンスベクトルに含まれる 1 ビットをローカルコリジョンを用いてメッセージ差分に展開する際、正のメッセージ差分から派生するローカルコリジョンと、負のメッセージ差分から派生するローカルコリジョンの二種類のバリエーションを取り得る。このバリエーションの違いにより、同一のディスターバンスベクトルから複数のメッセージ差分が導出されることになる。なお (d) については、本質的に差がない場合、あるいは差分攻撃を効率化する等の理由により、探索範囲を出来る限り広くしておくため、符号を未定の状態 (符号無しの状態) とすることも許容され得る。また、メッセージ差分の符号を決定する際には、各ローカルコリジョンを成立させるために必要なメッセージコンディションに矛盾しないように決定する必要がある。

5.1.1. 非線形パスに関する考察

非線形パスは、コリジョン探索を実際に行なう上で必要となるものである。文献[69][75]ではそれぞれ SHA-0, SHA-1 に対する非線形パスの導出結果が記載されているが、それらは手作業で数ヵ月かけて導出されたことが知られており、差分パスの導出法についての記載はない。その一方で、マルチブロックコリジョンを仮定したコリジョン探索計算量の詳細な見積り、ならびに攻撃アルゴリズムの構成には、非線形パス構築の自動化は必須であると考えられる。そのような背景から、これまでに文献[83][85][86][87][88]において、非線形パス導出の自動化に関する検討がなされている。

文献[86][87][88]では、ローカルコリジョンを可能な限り適用する逆方向探索、メッセージ差分を展開する順方向探索、ならびに算術キャリーを組み合わせた結合探索の三種類の探索アルゴリズムを組み合わせた方法が提案され、文献[75]に記載された差分とは異なる非線形パスの導出に成功している。

文献[83]では、ステップ 12 からステップ 16 にかけて実行する〝フリービット優先探索〟ならびに 1 ステップから 11 ステップにかけて実行する〝局所最適化探索〟を組み合わせた探索方法が提案され、文献[84]では、本手法を用いた差分パスを用い、70 ステップの SHA-1 のコリジョンの導出に成功している。

5.1.2. 線形パスに関する考察

線形パスは、コリジョン探索計算量に直接影響を与えるため、出来る限り少ないコンディションで、差分パスを構成すべきである。線形パスは、ディスターバンスベクトルをローカルコリジョン通りに展開すれば得られるが、先に述べたように、メッセージ差分の符号の違いによる差から、ディスターバンスベクトルから得られる線形パスを成り立たさせるためのコンディションの個数、すなわち攻撃計算量は一意的ではない。

文献[82]では、このディスターバンスベクトルの展開をより精密化し、メッセージの符号によって、コンディションの個数が大きく変わることを注意しなくてはならない点を指摘している。

5.2. 差分パス生成ツールを実現する上で解決すべき課題抽出

本節では、前節までに抽出された差分パス生成ツールを実現する上で解決すべき課題についてまとめる。

5.2.1. 非線形パスに関する課題

(1) 与えられた全てのディスターバンスベクトルに対し非線形パスが存在するとは限らない
(2) 非線形パスを構成することは（線形パスの構成と比較して）難しい
(3) 構成された非線形差分パスがコリジョン探索攻撃に有効とは限らない

5.2.2. 線形パスに関する課題

(1) メッセージ差分の符号の決め方
(2) メッセージコンディションの個数の評価
(3) メッセージフリーダムの個数
(4) 線形パスと非線形パスの境目におけるローカルコリジョンの扱い

6. 算術演算をベースとするハッシュ関数に対する攻撃計算量に関する調査

本節では、差分攻撃に基づく攻撃計算量について明らかにするため、攻撃を効率的に適用するための条件に関して調査を行う。

6.1. モディフィケーション技術の適用可能性を効率的に判定するための条件

本節では、メッセージモディフィケーション技術の適用可能性を効率的に判定する技術として、ベーシックモディフィケーション、ならびにアドバンストメッセージモディフィケーションについて既存文献に記載されている内容をまとめる。

与えられたディスターバンスベクトルに対して構成された差分パスを成立させるためには、複数の条件式が成立する必要があるが、このうち、メッセージに対して付加される条件式をメッセージコンディション、ステップの入出力に対する条件式をチェイニングバリアブルコンディションと呼ぶ。なお本節において、単にコンディションという場合は、チェイニングバリアブルコンディションを意味することとする。

文献[48][59][69]において、このコンディションを高い確率で成立させる為の方法として、メッセージモディフィケーションと呼ばれる方法が提案されている。メッセージモディフィケーションは、ハッシュ関数のコリジョン探索おける計算量削減のために必須の技術である。メッセージモディフィケーションには、メッセージ拡大非適用ステップ(非線形パート)に付随するコンディションに対して適用する「ベーシックメッセージモディフィケーション」(図 11)とメッセージ拡大適用ステップ(線形パート)に付随するコンディションに対して適用する「アドバンストメッセージモディフィケーション」(図 12)がある。

ベーシックメッセージモディフィケーションは、図 11 のように、コンディションを満たさないステップにおいて、同じステップ内の関連するメッセージビットをダイレクトに変化させることによって、コンディションを満たすようにするテクニックである。ベーシックメッセージモディフィケーションは、メッセージ拡大非適用ステップに付随する全てのコンディションに対して適用可能であるため、メッセージ拡大非適用ステップのコンディション数は、ハッシュ関数のコリジョン探索計算量には含まれない。

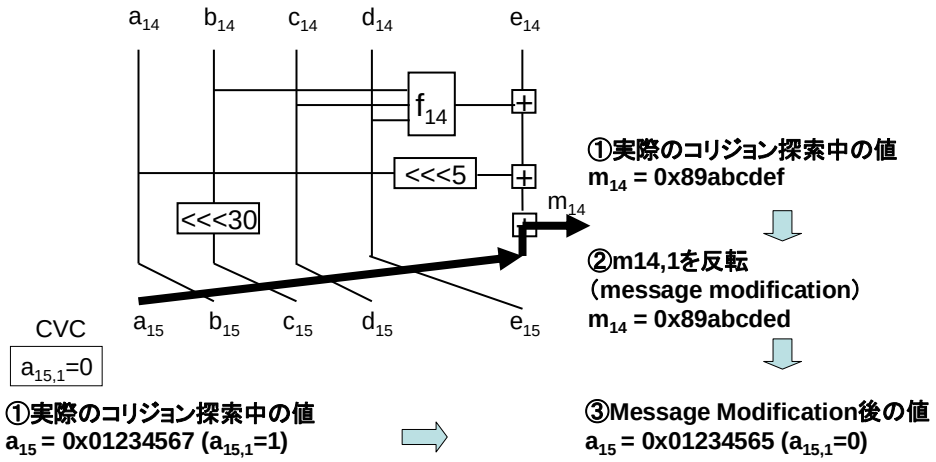

図 11. ベーシックメッセージモディフィケーション([95]より)

アドバンストメッセージモディフィケーションは、メッセージ拡大適用ステップに含まれるコンディションに対して実施される。ベーシックメッセージモディフィケーションと同様、コンディションを満たさないステップにおいて、同じくメッセージビットを変化させることによって、コンディションを満たすようにするテクニックである。しかし、同じステップのメッセージを変化させようとした場合、メッセージ拡大の影響により、他のステップのメッセージビットを変化させる必要が生じる。この変化は他のステップの入出力にも影響を与え、既に満たしていたコンディションを満たさなくなる恐れがあるため、安直にメッセージを修正することができない。このように、メッセージ拡大適用ステップに付随するコンディションに対しては、他のコンディションの影響を考慮してモディフィケーションを行なう必要があることから、ベーシックメッセージモディフィケーションと区別し、アドバンストメッセージモディフィケーションと呼ばれる手法を適用する。

アドバンストメッセージモディフィケーションとしてはいくつかの方法が提案されている。これまで提案されている方法を表 9 にまとめる。

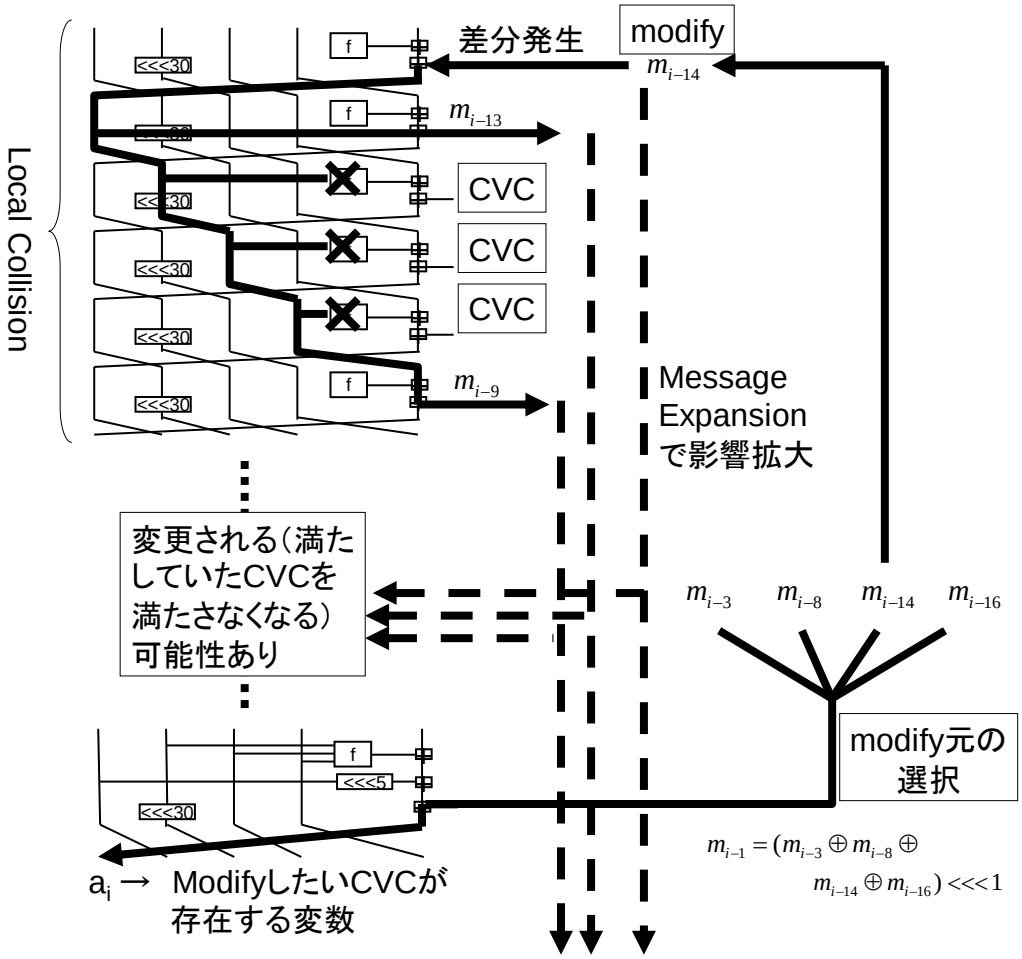

図 12. アドバンストメッセージモディフィケーション([95]より)

表 9. アドバンストメッセージモディフィケーション一覧

文献	手法	コメント
[48][59]	キャンセルモディフィケーション	SHA-1 の 19 ステップまで
[69]	プロパゲーションモディフィケーション	SHA-1 の 21 ステップまで
[62]	サブマリンメッセージモディフィケーション	SHA-1 の 25 ステップまで
[90]	ブーメラン法	条件次第ではステップ 25 以降も可能
[95]	バニッシングメッセージモディフィケーション	条件次第ではステップ 25 以降も可能
[91]	グレブナー基底を用いた方法	モディフィケーションの一般化(概念のみ)

6.2. 攻撃計算量を正確に見積もるためのツールが満たすべき要件

文献[75]をベースとしたコリジョン探索攻撃を行なう場合、マルチブロックコリジョン、すなわち複数のブロックを繋げて、差分パスを成立させることにより、コリジョンを生成させるという技術を用いることから、コリジョン探索攻撃計算量としては、ブロック数も考慮に入れるべきであると考えられる。すなわち、以下の式が成り立つ。

数式 3. コリジョン探索計算量の概算値

$$\text{全探索攻撃計算量} = \Sigma \text{ 各ブロックの探索計算量}$$

特に各ブロックの探索計算量が各々等しいと仮定した場合、右辺は次の通りとなる。

数式 4. コリジョン探索計算量の概算値の右辺

$$\text{各ブロックの探索計算量} \times \text{ブロック数}$$

続いて、各ブロックのコリジョン探索を行なうアルゴリズムは次の通りである。

各ブロックのコリジョン探索アルゴリズム

1. 全てのメッセージコンディションを満たすメッセージを作成
2. メッセージモディフィケーションを使用して、出来る限り多くのコンディションを満たすようメッセージを修正
3. 残りのコンディションを全て満たし、求める差分パスが成立するまでメッセージの取り直し、1 から繰り返す。

各コンディションが成立する確率を 1/2 とし、各コンディションが独立であると仮定した場合、上記攻撃アルゴリズムの計算量は、次の式で表せる。

数式 5. 各ブロックにおけるコリジョン探索計算量

$$\text{コリジョン探索計算量} = 2^{(N-n)} \text{ 回}$$

ただし
N = 満たすべき全てのコンディションの数
n = モディフィケーションが適用可能なコンディションの数

上記より、攻撃計算量を見積もる上では、コリジョンを起こす差分パスを成立させる為に

満たすべきコンディションを導出し、その中から、モディフィケーション適用可能なコンディションを、判別することが重要である。モディフィケーションとしては、ベーシックメッセージモディフィケーションと、アドバンストメッセージモディフィケーションに区分されるが、ベーシックメッセージモディフィケーションについては、メッセージ拡大非適用ステップにおける全てのコンディションが、その対象となると考えても大きな問題は生じない[75]と考えられることから、本質的にはアドバンストメッセージモディフィケーションに関する適用可能性について判別することが、攻撃計算量の厳密な評価につながることが分かる。

ハッシュ関数 SHA-1 に対するアドバンストメッセージモディフィケーションの手法としては、文献[75]で提案された後、表 9 で示したように多くの改良が提案されている。これらのアドバンストメッセージモディフィケーションを適用する場合に注意すべき点として、いくつか挙げられる。

文献[76][77]では、マルチメッセージモディフィケーションの適用判定順序によって、モディフィケーション可能なコンディションの個数が変化することが指摘されており、文献 [95]では、判定順序、ならびにその判定法に関する提案がされている。探索計算量を見積もる場合、単一のコンディションについて、マルチメッセージモディフィケーションが適用可能であるかどうかを判定することも課題の一つではあるが、より本質的には、出来る限り多くのコンディションをモディフィケーションする手段を見付けることが重要であると考えられる。よって探索計算量を評価する上では、コンディションに対する判定順序を含めて適切に探索する必要がある。

通常コンディションの他に、「エクストラコンディション」と呼ばれるコンディションを付加し、このコンディションを成立させておく必要がある。このエクストラコンディションを適切に管理し、他のコンディションと矛盾ないようにしなければならない。また、エクストラコンディションを全て満たしておかなければ、アドバンストメッセージモディフィケーションを行なうことができないため、このエクストラコンディション自身のモディフィケーション計算量も考慮しなければならない。経験的にはステップ数が進んだ位置にあるコンディションほどモディフィケーションに必要なエクストラコンディションの個数が増加し、アドバンストメッセージモディフィケーションは困難となる。

メッセージモディフィケーションは、メッセージ拡大非適用ステップ内のメッセージビットを複数変化させることで実施するが、この範囲に含まれるメッセージビット数は、SHA-1 の場合 16 ステップ× 32 ビット と限りがある。そのため、この数を越えてメッセージモディフィケーションを行なうことは出来ない。

コリジョン探索において、ビットの値を自由に選択できるメッセージビットは「メッセージフリーダムビット」と呼ばれているが、コリジョン探索計算量が 2^K の時、メッセージフリーダムビットの個数が K 個以下となった場合、メッセージの全探索空間を尽くしたとしても、その差分パスに従ったコリジョンメッセージが得られないことを意味するため、

コリジョン探索を行なう上で無意味である。この点は、差分パス探索でも指摘されている[83]が、メッセージモディフィケーションにおいても同様に考慮すべき課題である。

6.3. 攻撃計算量を見積もるツールを実現する上で解決すべき課題抽出

本節では、前節において考察した、攻撃計算量を見積もるツールを実現する上での課題を抽出する。

前節でも述べた通り、攻撃計算量を見積もる上では、コリジョンを起こす差分パスを成立させる為に満たすべきコンディションを導出し、その中から、モディフィケーション適用可能なコンディションを判別することが重要である。特に、アドバンストメッセージモディフィケーションに関する適用可能性について判別することが、攻撃計算量の厳密な評価につながることが導かれている。

つまり、アドバンストメッセージモディフィケーションがどれだけ多くのコンディションに適用できるかを判定することが本質的課題であるが、更に要件を具体化すると、考慮すべき課題は以下の通りとなる。

(1) マルチブロックコリジョンを考慮した計算量評価
(2) モディフィケーション適用判定順序
(3) エクストラコンディションに対する再帰的モディフィケーション判定
(4) メッセージフリーダムビット数と探索計算量に関する注意

7. 算術演算をベースとするハッシュ関数の安全性評価ツール仕様検討

前章までの調査結果を基に、算術演算をベースとしたハッシュ関数の安全性検証ツールを開発するための外部仕様の策定を行い、その適用可能性、実現性、課題等を分析する。

7.1. ツールの目的

例えばあるハッシュ関数を設計した場合、そのハッシュ関数が既存の攻撃に対してどの程度の耐性があるのかを見積もることは大変重要である。この「耐性」の客観的な評価指標の一つとして、コリジョン発見までの手間について、攻撃者の立場を想定して見積もった概算の計算量が挙げられる。そこで我々は、この計算量の見積もり評価を行うための安全性評価ツールの仕様検討を実施した。本ツールは、算術演算をベースとするハッシュ関数を対象とし、Wang の差分攻撃をベースとしたコリジョン探索を適用した際の探索計算量を、概算で見積もることを目的とする。

7.2. ツールの全体構成の検討

今回検討対象としている算術演算をベースとするハッシュ関数は図 1 のような構成をしているものとする。本節では、安全性検証ツールの外部仕様を定めるための検討を実施する。

7.2.1. 探索計算量の導出方針検討

コリジョン探索で必要となる計算量の概算値は、数式 5 を元に数式 6 で与えられることが知られている。

数式 6. コリジョン探索計算量

$$(\text{コリジョン探索計算量}) = 2^{(N-n)} \times B$$

ただし

N = 満たすべき全てのコンディションの数

n = メッセージモディフィケーションが適用可能なコンディションの数

B = マルチブロックコリジョンにおけるブロック数

数式 6 により、「コリジョン探索計算量」を見積もるためには、内部状態が満たすべき「コンディション」と「メッセージモディフィケーションが適用可能なコンディション」をそれぞれ導出すればよいことになる。ところでメッセージモディフィケーションには、「ベーシックメッセージモディフィケーション」と「アドバンストメッセージモディフィケーション」があり、数式 6 における「メッセージモディフィケーションが適用可能なコンディションの数」とは、両者のうちのいずれかが適用可能であるコンディションの数のことを言う。攻撃者の立場でのコリジョン探索計算量の概算見積もりにおいては、メッセージ拡大非適用ステップに存在する全ての「コンディション」には「ベーシックメッセージモディフィケーション」が適用可能と仮定しても大きな問題はない。そこで今回のツールの検討においては、メッセージ拡大適用ステップに存在するコンディションのみを安全性算出の対象とし、これらのコンディションの総数と、アドバンストメッセージモディフィケーションが適用可能なコンディションの数とを考慮して安全性を算出することにした。コリジョン探索計算量の考え方のイメージを図 13 に示す。

図 1 に示したような算術演算をベースとするハッシュ関数について、数式 6 にてコリジョン探索計算量を評価する方法を検討したところ、図 14 に示す 5 個のモジュールで評価を行えばよいことが判明した。各モジュールの仕様検討は次節以降にて行う。なお、図 14 には各モジュールの入出力が記載されているが、実装の際にはこれらの他に制御用のパラメータなども入出力を要する可能性がある。

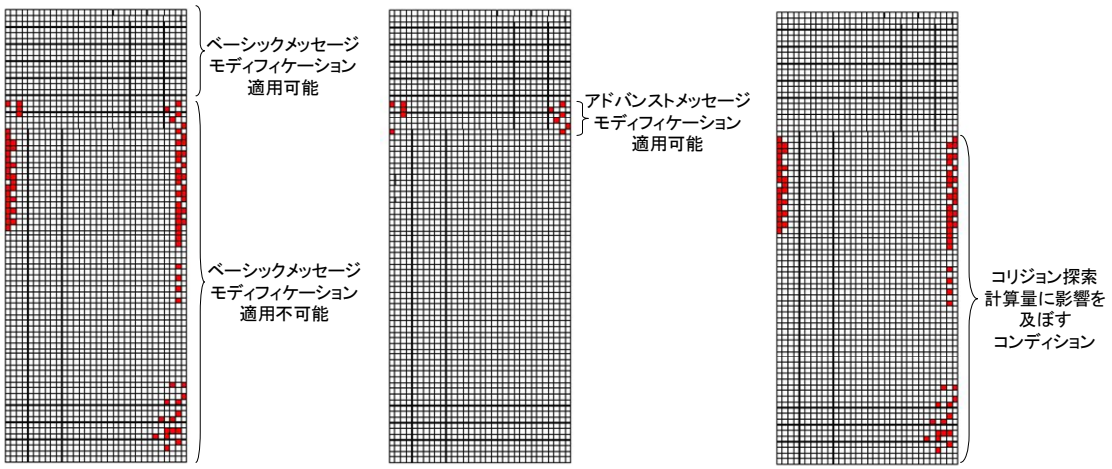

図 13. コリジョン探索計算量の考え方

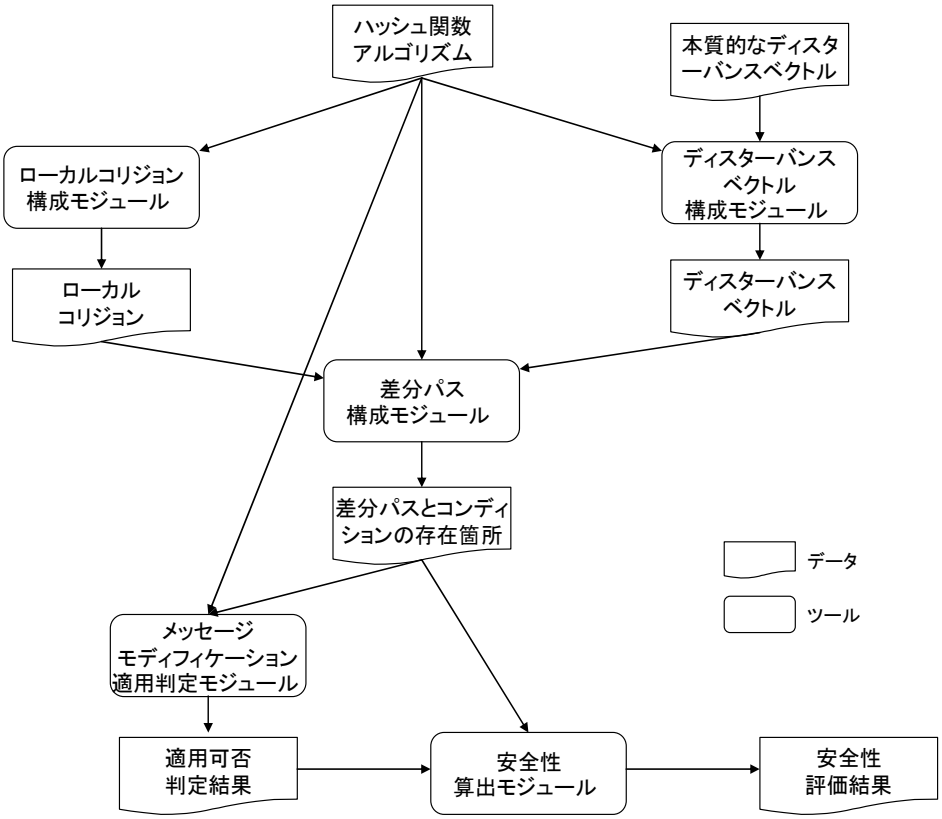

図 14. コリジョン探索計算量評価の処理のフロー

（上記の各モジュールには、上記のほかに制御パラメータなども入出力される）

7.2.2. ローカルコリジョンの導出方法検討

図 14 のフローにおけるローカルコリジョンの導出方針を検討する。多くのハッシュ関数のコリジョン探索では、「ローカルコリジョン」と呼ばれる差分パスを使用することが多い。これは、あるステップでメッセージに加わった差分が、少ないステップで吸収され、差分なしの状態になるような差分パスのことである。ローカルコリジョンは探索計算量見積もりの際に必須の情報であり、我々はローカルコリジョンの導出を「ローカルコリジョン構成モジュール」とすることにした。ローカルコリジョンは評価対象のハッシュ関数アルゴリズムが決定すれば検討可能と考えられる。ローカルコリジョン構成モジュールの入出力を図 15 に示す。

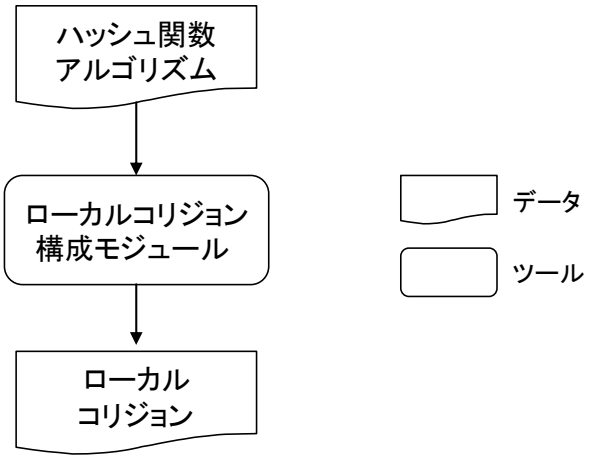

図 15. ローカルコリジョン構成モジュールの入出力

7.2.3. ディスターバンスベクトルの導出方針検討

図 14 のフローにおけるディスターバンスベクトルの導出方針を検討する。多くのハッシュ関数ではメッセージ拡大という処理があり、ブロック長分の入力メッセージに対して拡大処理が行われる。このため、ローカルコリジョンを成立させるためのメッセージ差分をあるステップに与えると、メッセージ拡大によって別のステップにもローカルコリジョンを生じることが多い。そこで一般的には、入力メッセージ空間と同じだけのサイズを持つ「本質的なディスターバンスベクトル」をまず生成し、それに対してメッセージ拡大と同等の変換処理を行ってディスターバンスベクトル全体を生成する処理が行われている。我々はこの処理を「ディスターバンスベクトル構成モジュール」とすることにした。「本質的なディスターバンスベクトル」は、評価対象のハッシュ関数アルゴリズムが決定し、そのメッセージ空間が決定すれば導出することが可能と考えられる。ディスターバンスベクトル構成モジュールの入出力を図 16 に示す。

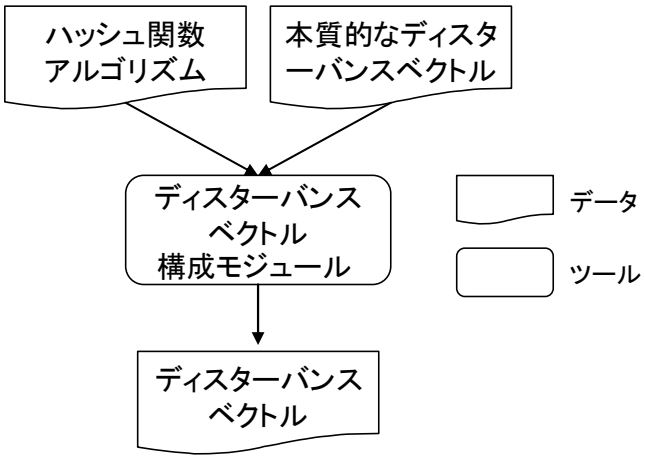

図 16. ディスターバンスベクトル構成モジュールの入出力

7.2.4. 内部状態が満たすべきコンディションの導出方針検討

図 14 のフローにおける差分パスの構成方法を検討する。コリジョン探索計算量を評価するためには、内部状態が満たすべきコンディションの数、及び、その存在位置（ビット位置）の情報が必要となる。ここで言う「コンディション」とは、事前に構築した差分パスを高確率で満たさせるための条件のことである。攻撃者の立場でのコリジョン探索計算量見積もりにおいては、差分パスはローカルコリジョンの組み合わせで構築されていると仮定して概ね問題ないと考えられる。つまり、内部状態が満たすべきコンディションの数を導出するためには、ディスターバンスベクトル（＝ローカルコリジョンの開始地点の情報）を元に、「コンディションの存在位置」を導出する処理を行う必要があると言い換えられる。我々はこの処理を「差分パス構成モジュール」とすることにした。差分パス構成モジュールの入出力を図 17 に示す。

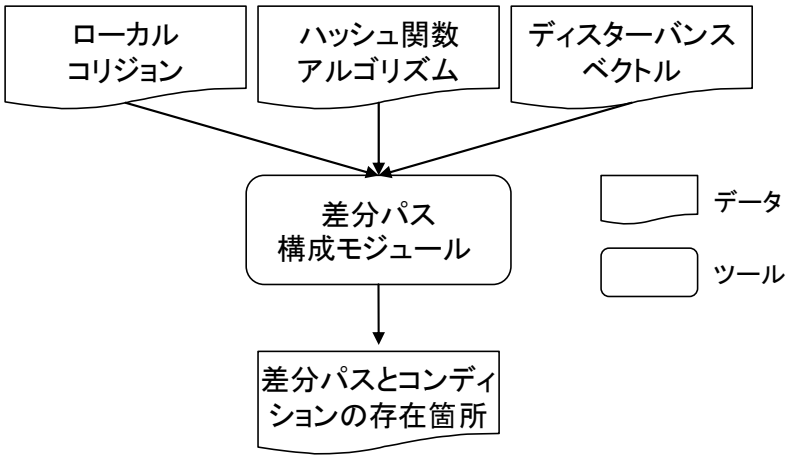

図 17. 差分パス構成モジュールの入出力

7.2.5. メッセージモディフィケーション適用判定方針検討

図 14 のフローにおけるメッセージモディフィケーション適用判定の検討を行う。7.2.1 節で説明した通り、今回の検討においては、アドバンストメッセージモディフィケーションのみを扱う。アドバンストメッセージモディフィケーションが適用可能かどうかの判定を厳密に行うためには、メッセージ拡大非適用ステップの「差分パス」、「コンディション」、及び、「メッセージフリーダム」を扱う必要がある。しかし今回は攻撃者の立場での計算量の概算見積もりであるため、これらは考慮しないことにした。(図 13 参照)。これにより、より多くのコンディションについて「アドバンストメッセージモディフィケーションが適用可能」と判定され、探索計算量がより少なく見積もられる可能性が否定できないが、、攻撃計算量の概算見積もりという観点では大きな問題はないと判断した。なおこの処理では、判定対象のコンディションを 1 個入力し、それに対するモディフィケーションの適用可否を判定する。従って全体の評価を行うためには、コンディション 1 個入力して判定する処理を複数のコンディションに対して繰り返し実行する方法が考えられる。ただし、得られる概算計算量評価値は、コンディションの判定順序（入力順序）に依存して変化する。このため、攻撃者にとって最も都合の良い（計算量の少ない）データを得るためには、判定順序を取りうる全バリエーションについて実行する必要がある。しかし、バリエーションの数は大変多い可能性があるため、効率の良い枝刈りが求められる。

我々はこの判定を「メッセージモディフィケーション適用判定モジュール」とすることにした。メッセージモディフィケーション適用判定モジュールの入出力を図 18 に示す。

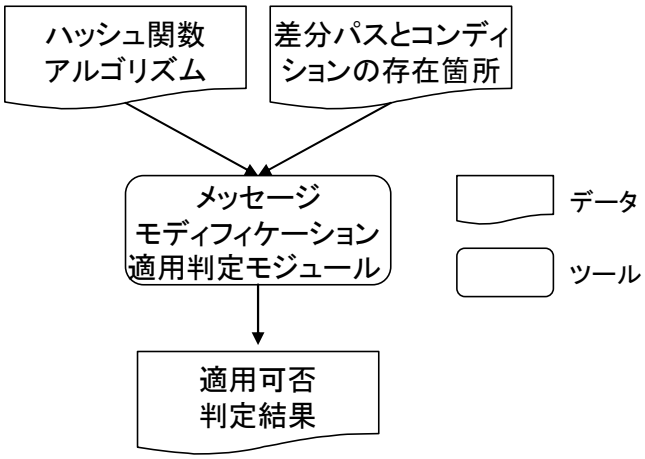

図 18. メッセージモディフィケーション適用判定モジュールの入出力

7.2.6. 安全性評価値算出方針検討

図 14 のフローにおける安全性算出手法について検討する。ここでいう安全性評価とは攻撃者の立場でのコリジョン探索計算量の概算値評価のことであり、7.2.1 節で説明した通り、数式 6 で与えられる。我々はこの処理を「安全性算出モジュール」とすることにした。安全性算出モジュールの入出力を図 19 に示す。

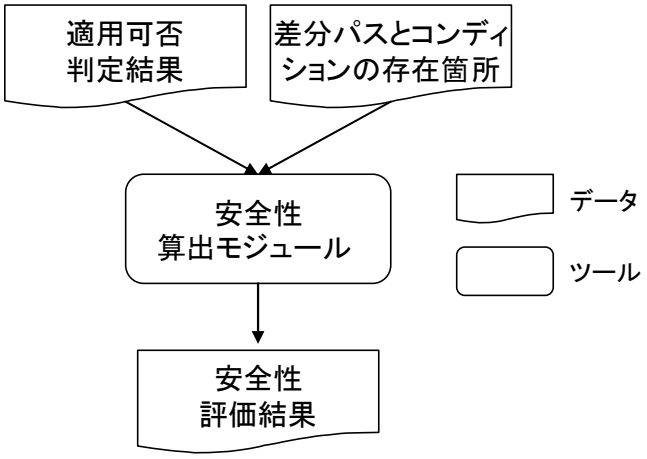

図 19. 安全性算出モジュールの入出力

7.3. 各モジュールの仕様

本節では、7.2 節で検討した 5 個のモジュールそれぞれについての仕様を説明し、実際のハッシュ関数での評価を検討する。7.2 節での検討により、これらのモジュールは以下の前提条件において計算量の概算値を計算することを目的としている。

前提条件

i. 複数ブロックでのコリジョン探索を前提とする。
ii. i の前提条件に基づき、差分パスは 1 ブロックでコリジョンさせるためのものではなく、各ブロックについて高い確率で成立するものを考慮する。
iii. 計算量の概算値は、i の前提条件における B ブロック分の値とする。
iv. 計算量の概算値は、攻撃者の立場で見積もる。
v. メッセージ拡大非適用ステップの差分パス（非線形パス）、及び、コンディションは考慮しない。
vi. 全コンディションに関する式全体の厳密な評価による矛盾判定は行わない。

7.3.1. ローカルコリジョン構成モジュール

概要：

入力されたハッシュ関数アルゴリズムについて、最小のステップ数でローカルコリジョンとなるような差分パスと、それを成立させるためのメッセージ差分、コンディションを出力する。

入力：

- ハッシュ関数のアルゴリズム

出力：

- 最小のステップ数でローカルコリジョンとなるようなメッセージの差分
- 上記ローカルコリジョンを成立させるために内部状態が満たすべきコンディションとメッセージコンディション

考察：

3.3 節で説明した通り、このモジュールは現状、人為的に実行する必要がある。計算機等でも実行可能な汎用的なアルゴリズムがあれば、より高速に処理可能となる。しかし本モジュールは、ハッシュ関数 1 種につき一般的には 1 回のみ実行するため、この問題は深刻ではないと考えられる。

適用例：

SHA-1 に適用した場合、表 10 のような結果が得られる。また、表 10 で示すローカルコリジョンを満たす例は図 2 に示すものと同じである。

表 10. ローカルコリジョン構成モジュールの適用結果例

ステップ	出力差分値 (差分パス)	メッセージ差分	内部状態が満たすべきコンディション
i	$(2^j, 0, 0, 0, 0)$	2^j	$a_{i,j} = 0$
i+1	$(0, 2^j, 0, 0, 0)$	$-2^{(j+5) \bmod 32}$	
i+2	$(0, 0, 2^{(j+30) \bmod 32}, 0, 0)$	-2^j	ラウンド 1: $a_{i-1,(j-30) \bmod 32} = 1$, $a_{i-2,(j-30) \bmod 32} = 0$ ラウンド 2: $a_{i-1,(j-30) \bmod 32} = a_{i-2,(j-30) \bmod 32}$ ラウンド 3: $a_{i-1,(j-30) \bmod 32} = a_{i-2,(j-30) \bmod 32} = 1$ ラウンド 4: $a_{i-1,(j-30) \bmod 32} = a_{i-2,(j-30) \bmod 32}$
i+3	$(0, 0, 0, 2^{(j+30) \bmod 32}, 0)$	$-2^{(j+30) \bmod 32}$	ラウンド 1: $a_{i+1,(j+30) \bmod 32} = 1$ ラウンド 2: $a_{i+1,(j+30) \bmod 32} = a_{i-1,j}$ ラウンド 3: $a_{i+1,(j+30) \bmod 32} = a_{i-1,j} = 1$ ラウンド 4: $a_{i+1,(j+30) \bmod 32} = a_{i-1,j}$
i+4	$(0, 0, 0, 0, 2^{(j+30) \bmod 32})$	$-2^{(j+30) \bmod 32}$	ラウンド 1: $a_{i+2,(j+30) \bmod 32} = 0$ ラウンド 2: $a_{i+2,(j+30) \bmod 32} = a_{i+1,j}$ ラウンド 3: $a_{i+2,(j+30) \bmod 32} = a_{i+1,j} = 1$ ラウンド 4: $a_{i+2,(j+30) \bmod 32} = a_{i+1,j}$
i+5	$(0, 0, 0, 0, 0)$	$-2^{(j+30) \bmod 32}$	

7.3.2. ディスターバンスベクトル構成モジュール

概要：

算術差分をベースとするハッシュ関数では、入力されたメッセージに対して、メッセージ拡大処理を行うことが多い。一方、ローカルコリジョンは、メッセージの差分からスタートしメッセージの差分に吸収される。このため、ハッシュ関数内のある場所にローカルコリジョンを発生するためのメッセージ差分をセットすると、メッセージ拡大の影響で他の場所にもローカルコリジョンが発生する。本モジュールでは、メッセージ拡大前のローカルコリジョンのスタート地点（ビット位置）を”1”で表現したものを「本質的なディスターバンスベクトル」と呼び、それを入力として受け取る。そしてモジュール内部において、メッセージ拡大適用後のディスターバンスベクトルを計算して出力する。

入力：

- ハッシュ関数のアルゴリズム
- 本質的なディスターバンスベクトル（7.2.3 節参照）

出力：

- ディスターバンスベクトル

考察：

本モジュールの入力である「本質的なディスターバンスベクトル」には非常に多くのバリエーションが存在する。このため、取りうる全パターンについて本モジュールを実行しようとすると、多くの計算時間・メモリを要することとなる。従って、「本質的なディスターバンスベクトル」の内、コリジョン探索の計算量が少なくなりそうなものを選択するのが現実的な解決策と考えられる。この選択方法は本モジュール実行前に検討する必要があるが、Wang の論文などに参考にすべき情報が記載されている。なお、本モジュール内部では、コンディションの数の概算値の見積もりを実行することも可能である。しかし、別のモジュールで実行することも可能であり、この見積もり自体を実施する必要があるかどうかという点を含めて実装の際に検討を要する。

制御パラメータとしては、探索範囲の情報などが考えられる。

適用例：

SHA-1 に表 11 のような入力を与えた場合、表 12 のような結果が得られる。

表 11. ディスターバンスベクトル構成モジュール適用例（入力データ）

ステップ	本質的なディスターバンスベクトル(“1”のビットがローカルコリジョンのスタート地点)	ステップ	本質的なディスターバンスベクトル(“1”のビットがローカルコリジョンのスタート地点)
1	0x40000001	9	0x00000002
2	0x00000002	10	0x00000002
3	0x00000002	11	0x00000000
4	0x80000002	12	0x00000000
5	0x00000001	13	0x00000001
6	0x00000000	14	0x00000000
7	0x80000001	15	0x80000002
8	0x00000002	16	0x00000002

表 12. 表 11 の入力に対するディスターバンスベクトル構成モジュールの適用結果例

ステップ	ディスターバンスベクトル	ステップ	ディスターバンスベクトル	ステップ	ディスターバンスベクトル	ステップ	ディスターバンスベクトル
1	0x40000001	21	0x00000003	41	0x00000000	61	0x00000000
2	0x00000002	22	0x00000000	42	0x00000000	62	0x00000000
3	0x00000002	23	0x00000002	43	0x00000002	63	0x00000000
4	0x80000002	24	0x00000002	44	0x00000000	64	0x00000000
5	0x00000001	25	0x00000001	45	0x00000002	65	0x00000004
6	0x00000000	26	0x00000000	46	0x00000000	66	0x00000000
7	0x80000001	27	0x00000002	47	0x00000002	67	0x00000000
8	0x00000002	28	0x00000002	48	0x00000000	68	0x00000008
9	0x00000002	29	0x00000001	49	0x00000002	69	0x00000000
10	0x00000002	30	0x00000000	50	0x00000000	70	0x00000000
11	0x00000000	31	0x00000000	51	0x00000000	71	0x00000010
12	0x00000000	32	0x00000002	52	0x00000000	72	0x00000000
13	0x00000001	33	0x00000003	53	0x00000000	73	0x00000008
14	0x00000000	34	0x00000000	54	0x00000000	74	0x00000020
15	0x80000002	35	0x00000002	55	0x00000000	75	0x00000000
16	0x00000002	36	0x00000002	56	0x00000000	76	0x00000000
17	0x80000002	37	0x00000000	57	0x00000000	77	0x00000040
18	0x00000000	38	0x00000000	58	0x00000000	78	0x00000000
19	0x00000002	39	0x00000002	59	0x00000000	79	0x00000028
20	0x00000000	40	0x00000000	60	0x00000000	80	0x00000080

7.3.3. 差分パス構成モジュール

概要：

コリジョン探索計算量の攻撃者の立場での概算評価で本質的に必要となる、ベーシックメッセージモディフィケーションが適用できないステップにおける差分パスを導出する。この差分パスはローカルコリジョンの組み合わせのみから成立すると仮定する。また、符号は最悪ケースでの評価では考慮する必要がないと考えられるため、考慮しない。

入力：

- ハッシュ関数のアルゴリズム
- ディスターバンスベクトル
- ローカルコリジョンとなるようなメッセージの差分とそのときに内部状態／メッセージが満たすべきコンディション

出力：

- 符号無し差分パス(ベーシックメッセージモディフィケーションが適用できないステップ)
- コンディション、もしくはコンディションが存在するビット位置の情報

考察：

ある差分パスについて、それを高確率で成立させるためのコンディションには複数のバリエーションが存在し、本モジュールではその中の１バージョンを出力する。しかし、どのバージョンを採用すると総合的な計算量が少なくなるかという点は未だ不明である。

また、「ベーシックメッセージモディフィケーションが適用可能なステップ」での差分パスの構築の困難性から、「ベーシックメッセージモディフィケーションが適用不可能なステップ」と「ベーシックメッセージモディフィケーションが適用可能なステップ」の境目付近にて、ローカルコリジョンの組み合わせによる差分パスが構成できないこともある。どのような条件で構成できなくなるか不明のため、本モジュールではこの点も考慮しないこととし、「ベーシックメッセージモディフィケーションが適用不可能なステップ」は全てローカルコリジョンの組み合わせで差分パスが構成されるとの前提で処理を行う。さらに「ベーシックメッセージモディフィケーションが適用不可能なステップ」に関連する内部状態（例えばSHA-1のステップ17における，a_{16}，a_{15}，a_{14}，a_{13}，a_{12}など）についてもローカルコリジョンの組み合わせで差分パスが構成されているものとする。

制御パラメータとしては、処理対象ステップの情報、コンディションが省略可能なステップの情報（例えば最終ステップなど）などが考えられる。

適用例：

SHA-1について、表 12のディスターバンスベクトルを用いて表 10のローカルコリジョンを設定すると表 13、及び、表 14のような結果が得られる。

表 13. 表 12 のディスターバンスベクトルに対する差分パスの例

ステップ	出力差分値（符号無し）	ステップ	出力差分値（符号無し）	ステップ	出力差分値（符号無し）	ステップ	出力差分値（符号無し）
17	0x80000002	33	0x00000003	49	0x00000002	65	0x00000004
18	0x00000000	34	0x00000000	50	0x00000000	66	0x00000000
19	0x00000002	35	0x00000002	51	0x00000000	67	0x00000000
20	0x00000000	36	0x00000002	52	0x00000000	68	0x00000008
21	0x00000003	37	0x00000000	53	0x00000000	69	0x00000000
22	0x00000000	38	0x00000000	54	0x00000000	70	0x00000000
23	0x00000002	39	0x00000002	55	0x00000000	71	0x00000010
24	0x00000002	40	0x00000000	56	0x00000000	72	0x00000000
25	0x00000001	41	0x00000000	57	0x00000000	73	0x00000008
26	0x00000000	42	0x00000000	58	0x00000000	74	0x00000020
27	0x00000002	43	0x00000002	59	0x00000000	75	0x00000000
28	0x00000002	44	0x00000000	60	0x00000000	76	0x00000000
29	0x00000001	45	0x00000002	61	0x00000000	77	0x00000040
30	0x00000000	46	0x00000000	62	0x00000000	78	0x00000000
31	0x00000000	47	0x00000002	63	0x00000000	79	0x00000028
32	0x00000002	48	0x00000000	64	0x00000000	80	0x00000080

表 14. 表 12 のディスターバンスベクトルにおけるメッセージ差分の例

ステップ	メッセージ差分（符号無し）	ステップ	メッセージ差分（符号無し）	ステップ	メッセージ差分（符号無し）	ステップ	メッセージ差分（符号無し）
1	0x60000000	21	0x20000001	41	0x80000002	61	0x00000000
2	0xE000002A	22	0x20000060	42	0x80000000	62	0x00000000
3	0x20000043	23	0x80000001	43	0x80000002	63	0x00000000
4	0xB0000040	24	0x40000042	44	0x80000040	64	0x00000000
5	0xD0000053	25	0xC0000043	45	0x00000000	65	0x00000004
6	0xD0000022	26	0x40000022	46	0x80000040	66	0x00000080
7	0x20000000	27	0x00000003	47	0x80000000	67	0x00000004
8	0x60000032	28	0x40000042	48	0x00000040	68	0x00000009
9	0x60000043	29	0xC0000043	49	0x80000000	69	0x00000101
10	0x20000040	30	0xC0000022	50	0x00000040	70	0x00000009
11	0xE0000042	31	0x00000001	51	0x80000002	71	0x00000012
12	0x60000002	32	0x40000002	52	0x00000000	72	0x00000202
13	0x80000001	33	0xC0000043	53	0x80000000	73	0x0000001A
14	0x00000020	34	0x40000062	54	0x80000000	74	0x00000124
15	0x00000003	35	0x80000001	55	0x00000000	75	0x0000040C
16	0x40000052	36	0x40000042	56	0x00000000	76	0x00000026
17	0x40000040	37	0x40000042	57	0x00000000	77	0x0000004A
18	0xE0000052	38	0x40000002	58	0x00000000	78	0x0000080A
19	0xA0000000	39	0x00000002	59	0x00000000	79	0x00000060
20	0x80000040	40	0x00000040	60	0x00000000	80	0x00000590

表 15. 表 13 の差分パスに対するコンディションの例

ステップ	コンディション
17	$a_{17,1} = f_{16,1}$, $m_{17,4} \neq a_{17,31}$, $m_{17,6} \neq a_{17,1}$, $m_{18,29} \neq f_{18,29}$, $a_{19,1} = f_{18,1}$, $m_{21,29} \neq a_{17,31}$
18	$a_{15,31} \neq f_{19,29}$
19	$m_{19,6} \neq a_{19,1}$, $m_{20,29} \neq f_{20,29}$
20	
21	$a_{21,0} = m_{20,0}$, $a_{21,1} = f_{20,1}$, $m_{21,5} \neq a_{21,0}$, $m_{21,6} \neq a_{21,1}$, $m_{22,0} \neq f_{22,0}$, $m_{25,30} \neq a_{21,0}$
22	
23	$a_{23,1} = f_{22,1}$, $m_{23,6} \neq a_{23,1}$
24	$a_{24,1} = m_{23,1}$, $m_{23,30} \neq f_{23,30}$, $m_{24,6} \neq a_{24,1}$
25	$a_{25,0} = m_{24,0}$, $m_{24,1} \neq f_{24,1}$, $m_{24,30} \neq f_{24,30}$, $m_{25,5} \neq a_{25,0}$, $m_{29,30} \neq a_{25,0}$
26	$m_{25,1} \neq f_{25,1}$
27	$a_{27,1} = m_{26,1}$, $m_{27,6} \neq a_{27,1}$, $m_{26,0} \neq f_{26,0}$
28	$a_{28,1} = m_{27,1}$, $m_{27,30} \neq f_{27,30}$, $m_{28,6} \neq a_{28,1}$
29	$a_{29,0} = m_{28,0}$, $m_{28,1} \neq f_{28,1}$, $m_{28,30} \neq f_{28,30}$, $m_{29,5} \neq a_{29,0}$, $m_{33,30} \neq a_{29,0}$
30	$m_{29,1} \neq f_{29,1}$
31	$m_{30,0} \neq f_{30,0}$
32	$a_{32,1} = m_{31,1}$, $m_{31,30} \neq f_{31,30}$, $m_{32,6} \neq a_{32,1}$
33	$a_{33,0} = m_{32,0}$, $a_{33,1} = m_{32,1}$, $m_{32,30} \neq f_{32,30}$, $m_{33,5} \neq a_{33,0}$, $m_{33,6} \neq a_{33,1}$, $m_{37,30} \neq a_{33,0}$
34	$m_{33,1} \neq f_{33,1}$
35	$m_{34,0} \neq f_{34,0}$, $a_{35,1} = f_{34,1}$, $m_{35,6} \neq a_{35,1}$
36	$a_{36,1} = m_{35,1}$, $m_{35,30} \neq f_{35,30}$, $m_{36,6} \neq a_{36,1}$
37	$m_{36,1} \neq f_{36,1}$, $m_{36,30} \neq f_{36,30}$
38	$m_{37,1} \neq f_{37,1}$, $a_{38,3} \neq a_{37,3}$
39	$a_{39,1} = m_{38,1}$, $m_{39,6} \neq a_{39,1}$, $a_{39,1} \neq m_{40,1}$
40	$a_{40,31} \neq a_{38,1}$
41	$a_{41,31} \neq a_{40,1}$

42	$a_{42,\ 3} \neq a_{41,\ 3}$
43	$a_{43,\ 1} = m_{42,\ 1},\ m_{43,\ 6} \neq a_{43,\ 1}$
44	$a_{44,31} \neq a_{42,\ 1},\ a_{44,\ 3} \neq a_{43,\ 3}$
45	$a_{45,\ 1} = a_{43,\ 1},\ m_{45,\ 6} \neq a_{45,\ 1},\ a_{45,31} \neq a_{44,\ 1}$
46	$a_{46,31} \neq a_{44,\ 1},\ a_{46,\ 3} \neq a_{45,\ 3}$
47	$a_{47,\ 1} = a_{45,\ 1},\ m_{47,\ 6} \neq a_{47,\ 1},\ a_{47,31} \neq a_{46,\ 1}$
48	$a_{48,31} \neq a_{46,\ 1},\ a_{48,\ 3} \neq a_{47,\ 3}$
49	$a_{49,\ 1} = a_{47,\ 1},\ m_{49,\ 6} \neq a_{49,\ 1},\ a_{49,\ 1} \neq m_{50,\ 1},\ a_{49,31} \neq a_{48,\ 1}$
50	$a_{50,31} \neq a_{48,\ 1}$
51	$a_{51,31} \neq a_{50,\ 1}$
52	
53	
54	
55	
56	
57	
58	
59	
60	
61	
62	
63	
64	
65	$a_{65,\ 2} = m_{64,\ 2},\ m_{65,\ 7} \neq a_{65,\ 2},\ m_{66,\ 2} \neq f_{66,\ 2},\ m_{69,\ 0} \neq a_{65,\ 2}$
66	$m_{67,\ 0} \neq f_{67,\ 0}$
67	$a_{68,\ 3} = m_{67,\ 3},\ m_{68,\ 8} \neq a_{68,\ 3},\ m_{72,\ 1} \neq a_{68,\ 3},\ m_{68,\ 0} \neq f_{68,\ 0}$
68	$m_{69,\ 3} \neq f_{69,\ 3}$
69	$m_{70,\ 1} \neq f_{70,\ 1}$
70	$m_{71,\ 1} \neq f_{71,\ 1}$

71	$a_{71,\ 4} = m_{70,\ 4}$, $m_{71,\ 9} \neq a_{71,\ 4}$, $m_{75,\ 2} \neq a_{71,\ 4}$, $m_{72,\ 4} \neq f_{72,\ 4}$
72	$m_{73,\ 2} \neq f_{73,\ 2}$
73	$a_{73,\ 3} = m_{72,\ 3}$, $m_{73,\ 8} \neq a_{73,\ 3}$, $m_{77,\ 1} \neq a_{73,\ 3}$, $m_{74,\ 2} \neq f_{74,\ 2}$, $m_{74,\ 3} \neq f_{74,\ 3}$
74	$a_{74,\ 5} = m_{73,\ 5}$, $m_{74,10} \neq a_{74,\ 5}$, $m_{75,\ 1} \neq f_{75,\ 1}$, $m_{75,\ 5} \neq f_{75,\ 5}$
75	$m_{76,\ 1} \neq f_{76,\ 1}$, $m_{76,\ 3} \neq f_{76,\ 3}$
76	$m_{77,\ 3} \neq f_{77,\ 3}$
77	$a_{77,\ 6} = m_{76,\ 6}$, $m_{77,11} \neq a_{77,\ 6}$, $m_{78,\ 6} \neq f_{78,\ 6}$
78	$m_{79,\ 4} \neq f_{79,\ 4}$
79	$a_{79,\ 5} = m_{78,\ 5}$, $a_{79,\ 3} = a_{74,\ 5}$, $m_{79,\ 8} \neq a_{79,\ 3}$, $m_{79,10} \neq a_{79,\ 5}$
80	$a_{80,\ 7} = m_{79,\ 7}$

7.3.4. メッセージモディフィケーション適用判定モジュール

概要：

入力されたコンディションのビット位置について、アドバンストメッセージモディフィケーションが適用可能かどうかを判定する。各コンディションについて順次実行すれば、全体でいくつのコンディションにモディフィケーションが適用できるかが分かる。

入力：

- ハッシュ関数のアルゴリズム
- 適用判定対象のコンディションが存在するビット位置の情報
- 適用判定対象でないコンディションが存在するビット位置の情報

出力：

- メッセージモディフィケーションの適用判定結果

考察：

モディフィケーションが適用可能なコンディションの総数は、条件の判定順序に依存して変化すると考えられるため、判定順序の複数のバリエーションにて本モジュールを実行する必要がある可能性がある。

制御パラメータとしては、コンディションの判定順序、ステータス、処理対象ステップの情報、メッセージフリーダムなどが考えられる。

適用例：

SHA-1 について、表 15 のコンディションについて、$a_{21,1} = f_{20,1}$を「適用判定対象のコンディション」として表 16 のように入力した場合、表 17 のような結果が得られる。この例では「適用可能」という出力が得られる。「適用不可能」と判定される場合の例を表 18、表 19 に示す。

表 16. メッセージモディフィケーション適用判定モジュールの入力結果例（成功例）

適用判定対象のコンディション	$a_{21,\ 1} = f_{20,\ 1}$
適用判定対象でないコンディション	表 15 のコンディション

表 17. メッセージモディフィケーション適用判定モジュールの出力結果例（成功例）

判定結果	適用可能
モディフィケーション経路	a21, 1 → a20, 28 a20, 28 → a19, 23 a19, 23 → a18, 18 a18, 18 → a17, 13 a17, 13 → a16, 8 a16, 8 → m15, 8

表 18. メッセージモディフィケーション適用判定モジュールの入力結果例（失敗例）

適用判定対象のコンディション	$a_{17,\ 1} = a_{16,1}$
適用判定対象でないコンディション	$a_{16,28} = m_{15,28}$ $a_{12,30} = m_{12,30}$ $a_{15,28} = m_{14,28}$ $a_{14,30} = m_{13,30}$ $a_{13,30} = m_{12,30}$ $m_{13,0} = 0$ $m_{8,0} = 1$ $m_{2,0} = 1$ $m_{0,0} = 0$

表 19. メッセージモディフィケーション適用判定モジュールの出力結果例（失敗例）

判定結果	適用不可能
モディフィケーション経路	なし

7.3.5. 安全性算出モジュール

概要：

入力された「内部状態が満たすべきコンディション」と「メッセージモディフィケーションの適用判定結果」から、コリジョン探索に必要な計算量の概算値を出力する。

入力：

- ベーシックメッセージモディフィケーションが適用できないステップに存在するコンディションの数
- 上記の内、メッセージモディフィケーションが適用可能なコンディションの数
- コリジョンが発生するためのブロック数

出力：

- コリジョン探索計算量

考察：

本モジュールの出力結果の精度は入力データに依存している。ディスターバンスベクトル構成モジュールの入力範囲をループさせながら、本モジュール実行までの処理を何回も実行する手段が考えられる。

制御パラメータとしては、処理対象ステップの情報などが考えられる。

適用例：

SHA-1 について、表 20 のようなデータを入力した場合、表 21 のような結果が得られる。

表 20. 安全性算出モジュールの入力データ例

コンディションの数	100
メッセージモディフィケーション適用可能なコンディションの数	30
ブロック数	2

表 21. 安全性算出モジュールの出力結果例

コリジョン探索計算量	2×2^{70}

7.4. ツールを用いたハッシュ関数の評価検討

本節では、前節までに検討した 5 個の安全性評価ツールが、実際のハッシュ関数の評価に使用可能かどうかを検討する。本検討では具体的に SHA-0、SHA-1、SHA-256 を対象として検討を行った。検討項目は以下の 3 点である。

- 適用可能性
  ツールを使用するための入力データが対象のハッシュ関数について準備可能かどうかを検討する。

- 実現性
  ツールを実行した際に現実的な時間で出力を得ることが可能かどうかを検討する。

- 課題
  上記 2 検討個目、及びそれ以外の事項において、ツールの実行に関連する課題を整理する。

本検討の結果を表 22、表 23、表 24、表 25、表 26 に示す。また、7.4.1 節、7.4.2 節、7.4.3 節にて、各ハッシュ関数の検討結果をより詳細に説明する。

表 22. 各ハッシュ関数に対するローカルコリジョン構成モジュールの適用性検討結果

検討対象項目	ハッシュ関数名		
	SHA-0	SHA-1	SHA256
適用可能性	○	○	○
実現性	○	○	○
課題	小	小	小

○：容易、△：やや困難、×：困難

表 23. 各ハッシュ関数に対するディスターバンスベクトル構成モジュールの適用性検討結果

検討対象項目	ハッシュ関数名		
	SHA-0	SHA-1	SHA-256
適用可能性	○	○	○
実現性	○	○	○
課題	小	小	小

○：容易、△：やや困難、×：困難

表 24. 各ハッシュ関数に対する差分パス構成モジュールの適用性検討結果

検討対象項目	ハッシュ関数名		
	SHA-0	SHA-1	SHA-256
適用可能性	○	○	○
実現性	○	○	○
課題	小	小	小

○：容易、△：やや困難、×：困難

表 25. 各ハッシュ関数に対するメッセージモディフィケーション適用判定モジュールの適用性検討結果

検討対象項目	ハッシュ関数名		
	SHA-0	SHA-1	SHA-256
適用可能性	○	○	○
実現性	△	△	△
課題	中	中	中

○：容易、△：やや困難、×：困難

表 26. 各ハッシュ関数に対する安全性算出モジュールの適用性検討結果

検討対象項目	ハッシュ関数名		
	SHA-0	SHA-1	SHA-256
適用可能性	○	○	○
実現性	○	○	○
課題	小	小	小

○：容易、△：やや困難、×：困難

7.4.1. SHA-0 の評価検討

SHA-0 は 1993 年に SHA として公開されたハッシュ関数である。1995 年に SHA-1 が公開された際に区別のためにそれまでの SHA を SHA-0 と表記するようになった。ブロック長は 512 ビット、ハッシュ値長は 160 ビットである。SHA-1 との差異はメッセージ拡大アルゴリズムにおけるローテーションシフトの有無であり、SHA-0 には 1 ビットローテーションシフトがない。SHA-0 には既にコリジョンが発見されている。

・ ローカルコリジョン構成モジュール

- 適用可能性
  本モジュールの入力は「ハッシュ関数アルゴリズム」であり、入力データの準備は容易である。このため、適用可能性は「容易」と考えられる。
- 実現性
  SHA-0 には、攻撃に有効なローカルコリジョンが存在することが知られており、本モジュールにて出力を得ることは可能である。このため実現性は「容易」と考えられる。
- 課題
  SHA-0 については有効なローカルコリジョンの存在が知られており、難易度は「小」とする。

・ ディスターバンスベクトル構成モジュール

- 適用可能性
  本モジュールの入力は「ハッシュ関数アルゴリズム」であり、入力データの準備は容易である。このため、適用可能性は「容易」と考えられる。
- 実現性
  SHA-0 のディスターバンスベクトルは、メッセージ拡大アルゴリズムを適用して導出することができる。このため実現性は「容易」と考えられる。
- 課題
  4.2 節で説明した通り、SHA-0 の場合、入力された「本質的なディスターバンスベクトル」には 2^{16} のバリエーションが存在すると考えられる。この数は大きくないため、課題の難易度は「小」と考えられる。

・ 差分パス構成モジュール

- 適用可能性

本モジュールの入力は、「ハッシュ関数アルゴリズム」、及び、ローカルコリジョン構成モジュールの出力とディスターバンスベクトル構成ツールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。

- 実現性

SHA-0 の場合、ディスターバンスベクトルに従ってローカルコリジョンをセットする処理は多くの計算量を要しない。従って実現性は「容易」と考えられる。

- 課題

SHA-0 においてはローカルコリジョンの組み合わせでの差分パスは構築可能であり、難易度は「小」と考えられる。

・ メッセージモディフィケーション適用判定モジュール

- 適用可能性

本モジュールへの入力は、「ハッシュ関数アルゴリズム」、及び、差分パス構成ツールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。

- 実現性

SHA-0 に関して、入力された「コンディションの存在箇所」情報から、その条件に対してモディフィケーションが適用できるかどうかを判定することは可能であると考えられる。しかし、判定順序のバリエーションの数が多い可能性があり、実現性は「やや困難」と考えられる。

- 課題

実現性の項目で説明した通り、判定順序のバリエーションの数が多い可能性があり、有効な判定順序の検討が必要となる可能性がある。このため課題の難易度は「中」と考えられる。

・ 安全性算出モジュール

- 適用可能性

本モジュールへの入力は、差分パス構成ツールの出力とモディフィケーション適用判定モジュールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。

- 実現性

SHA-0 に関して、数式 6 の計算を行うための計算量は少ないため、実現性は「容易」であると考えられる。

- 課題

大きな課題は見受けられないが、見積もり処理全体の高速化のために効率の良い実装が求められる。難易度は「小」と考えられる。

7.4.2. SHA-1 の評価検討

SHA-1 は 1995 年に公開されたハッシュ関数である。FIPS180-1、FIPS180-2 などとして公開されており、SSL(Secure Socket Layer)などでも標準的に使用されているハッシュ関数である。ブロック長は 512 ビット、ハッシュ値長は 160 ビットである。2008 年 2 月現在、SHA-1 のコリジョンは発見されていないが、安全とされる理論的な計算量である 2^{80} を下回る攻撃法が Wang らによって提案されている。

・ ローカルコリジョン構成モジュール

- 適用可能性
  本モジュールの入力は「ハッシュ関数アルゴリズム」であり、入力データの準備は容易である。このため、適用可能性は「容易」と考えられる。
- 実現性
  SHA-1 には、攻撃に有効なローカルコリジョンが存在することが知られており、本モジュールにて出力を得ることは可能である。このため実現性は「容易」と考えられる。
- 課題
  SHA-1 については有効なローカルコリジョンの存在が知られており、難易度は「小」とする。

・ ディスターバンスベクトル構成モジュール

- 適用可能性
  本モジュールの入力は「ハッシュ関数アルゴリズム」であり、入力データの準備は容易である。このため、適用可能性は「容易」と考えられる。
- 実現性
  SHA-1 のディスターバンスベクトルは、メッセージ拡大アルゴリズムを適用して導出することができる。このため実現性は「容易」と考えられる。
- 課題
  入力された「本質的なディスターバンスベクトル」には非常に多くのバリエーションが存在する。SHA-1 の場合、$2^{32\times16}$ パターン存在すると考えられるが、Wang の論文などに選択の際に参考にすべき情報が記載されているため、課題の難易度は「小」と考えられる。

・ 差分パス構成モジュール

- 適用可能性

  本モジュールの入力は、「ハッシュ関数アルゴリズム」、及び、ローカルコリジョン構成モジュールの出力とディスターバンスベクトル構成ツールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。
- 実現性

  SHA-1 の場合、ディスターバンスベクトルに従ってローカルコリジョンをセットする処理は多くの計算量を要しない。従って実現性は「容易」と考えられる。
- 課題

  SHA-1 においてはローカルコリジョンの組み合わせでの差分パスは構築可能であり、難易度は「小」と考えられる。

・ メッセージモディフィケーション適用判定モジュール

- 適用可能性

  本モジュールへの入力は、「ハッシュ関数アルゴリズム」、及び、差分パス構成ツールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。
- 実現性

  SHA-1 に関して、入力された「コンディションの存在箇所」情報から、その条件に対してモディフィケーションが適用できるかどうかを判定することは可能であると考えられる。しかし、判定順序のバリエーションの数が多い可能性があり、実現性は「やや困難」と考えられる。
- 課題

  実現性の項目で説明した通り、判定順序のバリエーションの数が多い可能性があり、有効な判定順序の検討が必要となる可能性がある。このため課題の難易度は「中」と考えられる。

・ 安全性算出モジュール

- 適用可能性

  本モジュールへの入力は、差分パス構成ツールの出力とモディフィケーション適用判定モジュールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。
- 実現性

  SHA-1 に関して、数式 6 の計算を行うための計算量は少ないため、実現性は「容易」であると考えられる。

- 課題

  大きな課題は見受けられないが、見積もり処理全体の高速化のために効率の良い実装が求められる。難易度は「小」と考えられる。

7.4.3. SHA-256 の評価検討

SHA-256 は SHA-1 の後継として 2000 年に提案されたハッシュ関数である。FIPS180-2 などとして公開されている。SHA-256、SHA-384、SHA-512 をまとめて SHA-2 と称することもある。ブロック長は 512 ビット、ハッシュ値長は 256 ビットである。2008 年 2 月現在、SHA-256 のコリジョンは発見されていない。

・ ローカルコリジョン構成モジュール

- 適用可能性
  本モジュールの入力は「ハッシュ関数アルゴリズム」であり、入力データの準備は容易である。このため、適用可能性は「容易」と考えられる。
- 実現性
  SHA-256 には、ローカルコリジョンが存在することが知られており、本モジュールにて出力を得ることは可能である。このため実現性は「容易」と考えられる。
- 課題
  SHA-256 についてはローカルコリジョンの存在が知られており、難易度は「小」とする。

・ ディスターバンスベクトル構成モジュール

- 適用可能性
  本モジュールの入力は「ハッシュ関数アルゴリズム」であり、入力データの準備は容易である。このため、適用可能性は「容易」と考えられる。
- 実現性
  SHA-256 のディスターバンスベクトルは、メッセージ拡大アルゴリズムを適用して導出することができる。このため実現性は「容易」と考えられる。
- 課題
  入力された「本質的なディスターバンスベクトル」には非常に多くのバリエーションが存在する。SHA-256 の場合、$2^{32\times16}$ パターン存在すると考えられるが、Wang の論文などに選択の際に参考にすべき情報が記載されているため、課題の難易度は「小」と考えられる。

・ 差分パス構成モジュール

- 適用可能性

本モジュールの入力は、「ハッシュ関数アルゴリズム」、及び、ローカルコリジョン構成モジュールの出力とディスターバンスベクトル構成ツールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。

- 実現性

SHA-256 の場合、ディスターバンスベクトルに従ってローカルコリジョンをセットする処理は多くの計算量を要しない。従って実現性は「容易」と考えられる。

- 課題

SHA-256 においてはローカルコリジョンの組み合わせでの差分パスは構築可能であり、難易度は「小」と考えられる。

・ メッセージモディフィケーション適用判定モジュール

- 適用可能性

本モジュールへの入力は、「ハッシュ関数アルゴリズム」、及び、差分パス構成ツールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。

- 実現性

SHA-256 に関して、入力された「コンディションの存在箇所」情報から、その条件に対してモディフィケーションが適用できるかどうかを判定することは可能であると考えられる。しかし、判定順序のバリエーションの数が多い可能性があり、実現性は「やや困難」と考えられる。

- 課題

実現性の項目で説明した通り、判定順序のバリエーションの数が多い可能性があり、有効な判定順序の検討を要する可能性がある。このため課題の難易度は「中」と考えられる。

・ 安全性算出モジュール

- 適用可能性

本モジュールへの入力は、差分パス構成ツールの出力とモディフィケーション適用判定モジュールの出力から得ることが可能であるため、適用可能性は「容易」と考えられる。

- 実現性

SHA-256 に関して、数式 6 の計算を行うための計算量は少ないため、実現性は「容易」であると考えられる。

- 課題

大きな課題は見受けられないが、見積もり処理全体の高速化のために効率の良い実装が求められる。難易度は「小」と考えられる。

8. References

* Fingerprint

[1] Max Gebhardt, Georg Illies, Werner Schindler,
"A Note on Practical Value of Single Hash Collisions for Special File Formats",
Proceedings of Cryptographic Hash Workshop, 2005.
http://csrc.nist.gov/groups/ST/hash/documents/Illies_NIST_05.pdf

[2] Magnus Daum, Stefan Lucks,
"The Story of Alice and her Boss, Hash functions and the Blind Passenger Attack",
Presented at the rump session of Eurocrypt 2005..
http://www.cits.rub.de/imperia/md/content/magnus/rump_ec05.pdf

[3] Marc Stevens, Arjen Lenstra, Benne de Weger,
"Predicting the winner of the 2008 US Presidential Elections using a Sony PlayStation 3", November 30, 2007.
http://www.win.tue.nl/hashclash/Nostradamus/

* Cryptanalysis of APOP and its variants (CHAP)

[4] J. Myers, M. Rose,
Post Office Protocol - Version 3,
Technical report, RFC1939(May 1996), RFC1957(June 1996), 1996.
http://www.ietf.org/rfc/rfc1939.txt

[5] W. Simpson
PPP Challenge Handshake Authentication Protocol (CHAP)
Technical report, RFC1994, August 1996.
http://www.ietf.org/rfc/rfc1994.txt

[6] Yu Sasaki, Go Yamamoto, Kazumaro Aoki,
Practical Password Recovery on an MD5 Challenge and Response

http://eprint.iacr.org/2007/101

[7] Gaan Leurent
Message Freedom in MD4 and MD5 Collisions: Application to APOP,
Proceedings of FSE2007, LNCS 4593, pp. 309-328. 2007.

[8] Lei Wang, Kazuo Ohta, Noboru Kunihiro,
Password Recovery Attack on Authentication Protocol MD4(Password||Challenge)
SCIS2008, 3A3-3, 2008.

* Cryptanalysis of HMAC and NMAC

[9] Mihir Bellare, R.Canetti and Hugo Krawczyk,
"Keying hash functions for message authentications",
Proceedings of CRYPTO1996, LNCS 1109, 1996.

[10] H. Krawczyk,M.Bellare, R.Canetti,
"HMAC: Keyed-hashing for message authentication",
RFC 2104, 1997.
http://www.ietf.org/rfc/rfc2104.txt

[11] American National Standard Institution,
"Keyed hash for message authentication code",
ANSI X9.71, 2000.

[12] NIST,
"The keyed-hash message authentication code (HMAC)",
FIPS PUB 198, March 2002.
http://csrc.nist.gov/publications/fips/fips198/fips-198a.pdf

[13] Mihir Bellare,
"New Proofs for NMAC and HMAC: Security without Collision-Resistance",
Proceedings of CRYPTO2006, LNCS 4117, pp. 602--620.

[14] Jongsung Kim, Alex Biryukov, Bart Preneel, Seokhie Hong,
On the Security of HMAC and NMAC Based on HAVAL, MD4, MD5, SHA-0 and SHA-1,
Proceedings of SCN 2006, LNCS 4116, pp. 242--256.

[15] Scott Contini, Yuqun Lisa Yin,
Forgery and Partial Key Recovery attacks on HMAC and NMAC using Hash Collisions
Proceedings of ASIACRYPT 2006, LNCS 4282, pp. 37-53., 2006.

[16] Pierre Alain Fouque, Gatan Leurent, Phong Nguyen,
Full Key-Recovery Attacks on HMAC/NMAC-MD4 and NMAC-MD5,
Proceedings of CRYPTO 2007, LNCS 4622, pp.13-30, 2007.

[17] Christian Rechberger, Vincent Rijmen,
On Authentication with HMAC and Non-Random Properties,
Proceedings of Financial Cryptography and Data Security LNCS 4886, pp.119-133, 2007.
http://eprint.iacr.org/2006/290

[18] Donghoon Chang, Jaechul Sung, Seokhie Hong and Sangjin Lee,
Improved Cryptanalysis of APOP-MD4 and NMAC-MD4 using New Differential Paths,
http://eprint.iacr.org/2008/048

* ANSI X9.62

[19] American National Standards Institute,
"Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)",
ANSI X9.62-1998, September, 1998.

[20] American National Standards Institute,
"Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)",
ANSI X9.62-2005, November, 2005.

* Cryptanalysis of X.509

[21] ITU-T,
"Recommendation X.509, The Directory - Authentication Framework."
http://www.itu.int/rec/T-REC-X.509/

[22] C. Adams, S. Farrell,
"Internet X.509 Public Key Infrastructure: Certificate Management Protocols",
RFC 2510, March 1999.
http://www.ietf.org/rfc/rfc2510.txt

[23] Arjen Lenstra, Xiaoyun Wang, Benne de Weger,
"Colliding X.509 Certificates", 2005.
http://eprint.iacr.org/2005/067

[24] Marc Stevens, Arjen Lenstra, Benne de Weger,
"Target Collisions for MD5 and Colliding X.509 Certificates for Different Identities", 2006.
http://eprint.iacr.org/2006/360

[25] Marc Stevens, Arjen Lenstra, Benne de Weger
"Chosen-prefix Collisions for MD5 and Colliding X.509 Certificates for Different Identities",
Proceedings of EUROCRYPT2007, LNCS4515, pp.1-22. 2007.

[26] Arjen Lenstra, Xiaoyun Wang, Benne de Weger,
Colliding X.509 Certificates for Different Identities
http://www.win.tue.nl/hashclash/TargetCollidingCertificates/
last modified March 17, 2006.

[27] Arjen Lenstra, Benne de Weger,
Colliding X.509 Certificates based on MD5-collisions
http://www.win.tue.nl/~bdeweger/CollidingCertificates/

last modified November 30, 2007.

* PKCS

[28] RSA Laboratories
Public-Key Cryptography Standards (PKCS)
http://www.rsa.com/rsalabs/pkcs

[29] Housley, R., Polk, W., Ford, W., and D. Solo,
"Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile",
RFC 3280, April 2002.
http://www.ietf.org/rfc/rfc3280.txt

[30] Arjen Lenstra, Benne de Weger,
"On the possibility of constructing meaningful hash collisions for public keys",
Proceedings of Information Security and Privacy 2005, LNCS 3574, pp.267-279, 2005

* SSL/TLS

[31] Alan O. Freier, Philip Karlton, Paul C. Kocher,
"The SSL Protocol Version 3.0",
Transport Layer Security Working Group, INTERNET-DRAFT, November, 1996.
http://wp.netscape.com/eng/ssl3/draft302.txt

[32] T. Dierks, C. Allen,
"The TLS Protocol Version 1.0",
RFC 2246, January 1999
http://www.ietf.org/rfc/rfc2246.txt

[33] T. Dierks, E. Rescorla
"The Transport Layer Security (TLS) Protocol Version 1.1",

RFC 4346, April, 2006.
http://www.ietf.org/rfc/rfc4346.txt

[34] Vlastimili Klima, Ondej Pokorny, Tomas Rosa
"Attacking RSA-based Sessions in SSL/TLS",
Cryptographic Hardware and Embedded Systems(CHES2003), LNCS 2779, pp.426--440 , 2003.

* Timestamp

[35] C.Adams, P.Chain, D.Pinkas, R.Zuccherato,
``Internet X.509 Public Key Infrastructure: Time-Stamp Protocol (TSP)",
RFC 3161, August 2001.
http://www.ietf.org/rfc/rfc3161.txt

[36] Tetsuya Izu, Takeshi Shimoyama, Masahiko Takenaka,
"How to Forge a Time-Stamp which Adobe's Acrobat Accepts",
Eleventh IMA International Conference on Cryptography and Coding IME-ICCC2007, pp.56-74, 2007.

* IPSEC

[37] R. Atkinson,
"Security Architecture for the Internet Protocol",
RFC 1825, August 1995.
http://www.ietf.org/rfc/rfc1825.txt

[38] R. Atkinson,
"IP Authentication Header",
RFC 1826, August 1995.
http://www.ietf.org/rfc/rfc1826.txt

[39] R. Atkinson,

"IP Encapsulating Security Payload (ESP)",
RFC 1827, August 1995.
http://www.ietf.org/rfc/rfc1827.txt

[40] P. Metzger and W. Simpson,
"IP Authentication using Keyed MD5",
RFC 1828, August 1995.
http://www.ietf.org/rfc/rfc1828.txt

[41] P. Karn, P. Metzger and W. Simpson,
"The ESP DES-CBC Transform",
RFC 1829, August 1995.
http://www.ietf.org/rfc/rfc1829.txt

[42] Kelly, S. and S. Frankel,
"Using HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512 With IPsec",
RFC 4868, May 2007.
http://www.ietf.org/rfc/rfc4868.txt

[43] P. Hoffman,
"Use of Hash Algorithms in Internet Key Exchange (IKE)",
RFC 4894, May 2007.
http://www.ietf.org/rfc/rfc4894.txt
(参考 http://www.rfcnews.jp/archives/2007/06/rfc_4894ikeipse.html

* Collision Attack of MD4

[44] Ronald L. Rivest,
"The MD4 Message Digest Algorithm", RFC1320, April 1992.
http://www.ietf.org/rfc/rfc1320.txt

[45] Bert den Boer, Antoon Bosselaers
“An Attack on the Last Two Rounds of MD4.”,
Proceedings of CRYPTO 1991. LNCS 576, pp.194--203, 1991.

[46] Hans Dobbertin,
“Cryptanalysis of MD4”,
Proceedings of Fast Software Encryption 1996, LNCS 1039, pp.53--69, 1996.

[47] Hans Dobbertin,
“Cryptanalysis of MD4”,
J. Cryptology 11(4), pp.253--271, 1998.

[48] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, Xiuyuan Yu,
“Cryptanalysis of the Hash Functions MD4 and RIPEMD”,
Proceedings of EUROCRYPT 2005, LNCS 3494, pp. 1--18, 2005.

[49] 岩崎 輝星、下山 武司,
"MD4 Collision Attack の差分パスおよび Sufficient Condition について",
SCIS2006, 4E2-1, 2006.

[50] Martin Schlaffer, Elisabeth Oswald,
“Searching for Differential Paths in MD4”,
Proceedings of FSE2006, LNCS 4047, pp. 242--261, 2006.

[51] Yu Sasaki, Lei Wang, Kazuo, Ohta, Noboru Kunihiro,
“New Message Difference for MD4”,
Proceedings of FSE2007, LNCS 4593, pp. 329-348. 2007.

* MD4 Second Pre-image Attack

[52] Hongbo Yu, Gaoli Wang, Guoyan Zhang, Xiaoyun Wang
"The Second-Preimage Attack on MD4",
Proceedings of Cryptology and Network Security 2005, LNCS 3810, pp.1-12, 2005.

[53] John Kelsey, Bruce Schneier,
"Second Preimages on n-bit Hash Functions for Much Less than 2^n Work",
November 2004.

http://eprint.iacr.org/2004/304

* Collision Attack of MD5

[54] Rivest, R., and S. Dusse, "The MD5 Message-Digest Algorithm",
MIT Laboratory for Computer Science and RSA Data Security, Inc.,
RFC 1321, April 1992.
http://www.ietf.org/rfc/rfc1321.txt

[55] Hans Dobbertin,
Cryptanalysis of MD5 Compress,
2006.
http://www-cse.ucsd.edu/~bsy/dobbertin.ps

[56] Hans Dobbertin, "The Status of MD5 After a Recent Attack",
CryptoBytes Volume 2, Number 2, pp.1,3-6, Summer 1996.

[57] 田島 直樹, 大川 晃弘, 金子 敏信
“MD5 の衝突に関する一検討”,
電子情報通信学会 ISEC, Vol.98, No.426, pp.1-7 1998.

[58] Bert den Boer, Antoon Bosselaers,
"Collision for the Compression Function of MD5",
Proceedings of EUROCRYPT1993, LNCS 765, pp. 293-304, 1994.

[59] Xiaoyun Wang, Xiuyuan Yu,
"How to Break MD5 and other Hash Functions",
Proceedings of EUROCRYPT2005 LNCS 3494, pp 19--35, 2005.

[60] John Black, Martin Cochran, Trevor Highland,
A Study of the MD5 Attacks: Insights and Improvements,
Proceedings of FSE2006, LNCS 4047, pp.262-277, 2006.

[61] V. Klima,

"Tunnels in Hash Functions: MD5 Collisions Within a Minute",
http://eprint.iacr.org/2006/105/

[62] Yusuke Naito, Yu Sasaki, Noboru Kunihiro, Kazuo Ohta,
"Improved Collision Attacks on MD4 and MD5",
IEICE Transactions E90-A(1) pp.36-47, 2007.

[63] Yu Sasaki, Yusuke Naito, Jun Yajima, Takeshi Shimoyama, Noboru Kunihiro, and Kazuo Ohta,
"How to Construct Sufficient Conditions for Hash Functions",
Proceedings of Vietcrypt2006, LNCS4341, pp. 243--259, 2006.

[64] 佐々木悠, 内藤祐介, 矢嶋純, 下山武司, 國廣昇, 太田和夫,
"How to Construct Sufficient Condition in Searching Collisions of MD5",
SCIS2006, 4E1-1, 2006.

[65] 矢嶋 純, 下山武司,
"MD5 のコリジョン探索および Sufficient Conditions について",
ISEC2005-78, pp.15-22, 2005.

[66] 矢嶋純, 内藤祐介, 佐々木悠, 下山武司, 國廣昇, 太田和夫,
"MD5 のコリジョン探索における差分パスの構築法について",
SCIS2006 4E1-2, 2006.

* Collision Attack of SHA-0

[67] Eli Biham, Rafi Chen,
"Near Collisions of SHA-0",
Proceedings of CRYPTO2004, LNCS 3152, pp.290--305, 2004.

[68] Eli Biham, Rafi Chen, Antonie Joux, Patrick Carribault, Christophe Lemuet, William Jalby,
"Collisions of SHA-0 and Reduced SHA-1",
Proceedings of EUROCRYPT2005, LNCS 3494, pp. 36-57, 2005.

[69] Xiaoyun Wang, Xiuyuan Yu, Yuqun Lisa Yin,
"Efficient Collision Search Attacks on SHA-0",
Proceedings of CRYPTO2005, LNCS3621, pp. 1-16. 2005.

[70] S. Manuel, T. Peyrin,
"Collisions on SHA-0 in one hour",
IPA Cryptographic Workshop 2007, 2007.

[71] Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Jun Yajima, Noboru Kunihiro and Kazuo Ohta,
"Improved Collision Search for SHA-0",
Proceedings of ASIACRYPT2006, LNCS 4284, pp. 21-36, 2006.

* Collision Attack of SHA-1

[72] NIST,
"Secure hash standard",
Federal Information Processing Standard, FIPS180, 1993.
http://csrc.nist.gov/publications/PubsFIPS.html

[73] NIST,
"Secure hash standard",
Federal Information Processing Standard, FIPS180-1, April 1995
http://csrc.nist.gov/publications/PubsFIPS.html

[74] NIST,
"Secure hash standard",
Federal Information Processing Standard, FIPS180-2, August 2002.
http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[75] Xiaoyun Wang, Yuqun Lisa Yin, Xiuyuan Yu,
"Finding Collisions in the Full SHA-1",

Proceedings of CRYPTO2005, LNCS 3621, pp.17-36, 2005.

[76] Xiaoyun Wang, A.C. Yao, F. Yao,
"Cryptanalysis on SHA-1 Hash Function",
Keynote Speech at CRYPTOGRAPHIC HASH WORKSHOP 2005.

[77] Xiaoyun Wang,
"Cryptanalysis of Hash functions and Potential Dangers",
Invited Talk at CT-RSA, 2006.

* Collision Attack of SHA-2

[78] Florian Mendel, Norbert Pramstaller, Christian Rechberger, Vincent Rijmen,
“Analysis of Step-Reduced SHA-256”,
Proceedings of FSE2006, LNCS 4047, pp.126--143, 2006

* Local Collision

[79] F. Chabaud, A.Joux,
"Differential Collisions in SHA-0",
CRYPTO'98, LNCS 1462, pp.56-71, 1998.

[80] F. Mendel, N.Pramstaller, C. Rechberger and V. Rijmen,
"The impact of carries on the complexity of collision attacks on SHA-1",
FSE 2006, LNCS 4047, pp.278--292, 2006

[81] 岩崎 輝星, 内藤 祐介, 矢嶋 純, 佐々木 悠, 下山 武司, 國廣 昇, 太田 和夫,
"Strategy for Selecting Disturbance Vector of SHA-1",
SCIS2007 1A1-3, 2007

[82] Jun Yajima, Terutoshi Iwasaki, Yusuke Naito, Yu Sasaki, Takeshi Shimoyama, Noboru Kunihiro, Kazuo Ohta,

"A Strict Evaluation Method on the Number of Conditions for the SHA-1 Collision Search",
To be appeared in ASIACCS08.

* Differential Path (non-linear part)

[83] Canni`ere,C.D., Rechberger, C.,
"Finding SHA-1 Characteristics: General Results and Applications",
ASIACRYPT2006, LNCS vol. 4284, pp.1--20, Springer 2006.

[84] Canni`ere,C.D., F. Mendel, C. Rechberger,
"Collisions for 70-step SHA-1: On the full cost of Collision Search",
SAC2007, LNCS 4876, pp56--73, 2007.

[85] P.Hawkes, M. Paddon, G.Rose.
"Automated search for round 1 differentials for SHA-1",
NIST Second Cryptographic HASH Workshop NIST August 2006.

[86] Jun Yajima, Yu Sasaki, Yusuke Naito, Terutoshi Iwasaki, Takeshi Shimoyama, Noboru Kunihiro, Kazuo Ohta,
"A New Strategy for Finding a Differential Path of SHA-1", ACISP 2007, LNCS 4586, pp.45--58, 2007.

[87] 佐々木 悠, 内藤 祐介, 矢嶋 純, 岩崎 輝星, 下山 武司, 國廣 昇, 太田 和夫,
"SHA-1 差分パス構築アルゴリズム",
SCIS2007 1A1-4, 2007.

[88] 矢嶋 純, 佐々木 悠, 岩崎 輝星, 内藤 祐介, 下山 武司, 國廣 昇, 太田 和夫,
"SHA-1 差分パス自動生成ツール",
SCIS2007 1A1-5, 2007.

* Message Modification

[89] Antoine Joux.
"Message Modification, Neutral Bits and Boomerangs: From Which Round Should we Start Counting in SHA?",
Keynote Speech at NIST SECOND CRYPTOGRAPHIC HASH WORKSHOP, Aug 2006.

[90] A.Joux, T. Peyrin,
"Hash Functions and (amplified) Boomerang Attack",
CRYPTO2007, pp. 244-263, 2007.

[91] Makoto Sugita, Mitsuru Kawazoe, Ludovic Perret, Hideki Imai,
“Algebraic Cryptanalysis of 58-Round SHA-1”,
Proceedings of FSE2007, LNCS 4593, pp. 349-365. 2007.

[92] V. Rijmen and E. Oswald,
"Update on SHA-1",
CT-RSA2005, LNCS 3376, pp 58-71. 2005.

[93] 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.

[94] 内藤祐介, 太田和夫, 國廣昇,
"ハッシュ関数のコリジョン探索の改良-新たな Advanced Message Modification の提案-",
SCIS2007,1A1-1, 2007.

[95] 矢嶋 純, 下山 武司,
SHA-1 のコリジョン探索における Message Modification 適用可否判定法
SCIS2008, 3A4-4, 2008.