Vol.2010-CSEC-50 No.11
2010/7/1

Quaternion（四元数）を応用した暗号に関する研究

門間 朋美、須藤 智寛、長瀬 智行、吉岡 良雄

本報告では、Quaternion（四元数）演算を基礎とするハッシュ関数 QHF を扱い、ハッシュ値生成や衝突耐性の向上などの比較を行った。より高度な衝突耐性を得るために可能性のある全ての攻撃について考察し、同サイズの異なる形式でのハッシュ値の生成と速度比較を行った。さらに、text ファイルの改竄に対するシミュレーションも行った。QHF に対しての攻撃確率も評価しており、衝突確率は n=128 で $1/2^{512}$ であり、要素長 n によってさらに低くすることも可能である。

Research on designing a New cryptography System based on Quaternion

Tomomi Monma†, Tomohiro Suto†, Tomoyuki Nagase†, and Yoshio Yoshioka†

This paper introduces a new paradigm for designing a new hash function based on quaternion algebra called Quaternion Hash Function *QHF*. Computer based experiments are conducted to compute hash values from input quaternion data by modifying an original text file. The modifications were done by adding, deleting and changing the contents of one-byte value. Since the quaternion has non-commutative feature among its multiplication's parameters, *QHF* is capable to resist hash collision.

1. はじめに

過去、多くの種類のハッシュ関数が考案･利用されてきた。その一例として、ロナルド･リベスト氏が考案したMD4およびその発展版のMD5が存在する。しかし、多くの研究者によってハッシュ関数に対する衝突攻撃が発見され、MD5やSHA-1などの脆弱性が指摘された。ハッシュ関数ではいかにして高い衝突耐性を得るかが問題となる。そのため、既存のハッシュ関数に対する改良や新型ハッシュ関数の設計によって従来のハッシュ関数よりも高い衝突耐性を得る研究が盛んに行われている。

本報告では Quaternion（四元数）を応用したハッシュ関数の新型アルゴリズムを提案する。今回提案する新型アルゴリズムは、Quaternion（四元数）の本来の使い方である回転行列を計算に含めずにQuaternion（四元数）の乗算の特性を活かしたアルゴリズムである。このような新型アルゴリズムを用いたハッシュ関数をQHF（Quaternion Hash Function）と呼ぶ。このQHFでは、ハッシュ関数の衝突耐性や0に対する処理を施してもハッシュ値が生成できることを明らかにする。

2. Quaternion(四元数)

Quaternion（四元数）は1843年にHamiltonが発見した複素数の自然な拡張であり、1個の実数成分と3 種類の超虚数成分からなる Hyper Complex（超複素数）の１つである。この四元数は以下の式で表される。

$$q = q_0 + iq_1 + jq_2 + kq_3$$

ここで、i、j、k は超虚数単位であり、以下のように定義されている。

$$ij = -ji = k$$
$$jk = -kj = i$$
$$ki = -ik = j$$
$$ii = jj = kk = -1$$

四元数の計算を通常の文字式のようにして扱う場合、四元数の実数成分をスカラー、超虚数成分をベクトルとすると式の単純化が可能である。

† 弘前大学大学院理工学研究科
Hirosaki University,

ⓒ 2010 Information Processing Society of Japan

Vol.2010-CSEC-50 No.11
2010/7/1

$V=(q_1,\ q_2,\ q_3)$
$q=(q_0,\ V)$

ここから四則演算・逆四元数を示すと、四元数

$p=(p_0, U)$、$q=(q_0,\ V)$ を用いて以下の式のように表現できる。

$$p\pm q=(p_0\pm q_0\ ,\ U\pm V)$$
$$pq=(p_0q_0-U\cdot V\ ,\ \ p_0V+q_0U+U\times V)$$
$$\frac{p}{q}=pq^{-1}=\frac{(p_0,\ U)(q_0,\ -V)}{\|q\|^2}=\frac{(p_0q_0-U\cdot V\ ,\ -p_0V+q_0U-U\times V)}{\|q\|^2}$$
$$q^{-1}=\frac{(q_0,\ -V)}{\|q\|^2}$$
$$qq^{-1}=q^{-1}q=1$$

ここで、$\|q\|=\sqrt{{q_0}^2+{q_1}^2+{q_2}^2+{q_3}^2}$ を四元数 q のノルムといい、$\|q\|=1$ のとき、q を単位 Quaternion という。

3. 四元数ハッシュ関数システムモデル

ハッシュ関数が生成する数値は四元数の各要素であるため、まず、data-quaternion processing と言うべき、入力データから四元数を作り出す前処理が必要である。前処理の最初の段階では、入力データを図１のようにブロック列として分割する。このとき、それらのサイズは 256 バイト、512 バイト、あるいはそれ以上などユーザによって自由に決めてよい。このブロック列を四元数列として扱うため、各ブロックを a_n, b_n, c_n, d_n のように等しい長さでさらに 4 分割する。

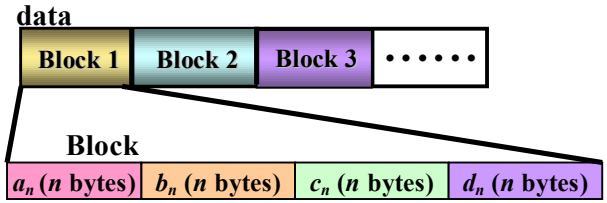


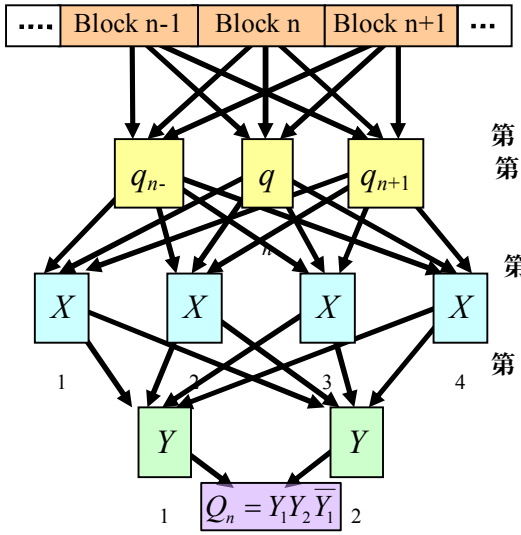
図１　データの切り出し

QHFの基本処理はブロック毎の四元数の乗算であるため、次式がベースである。

$$H=Q_1Q_2,\cdots,Q_n \qquad \cdots\cdots\cdots(1)$$

ここで、$Q_1, Q_2, \ldots, Q_N$ は入力データから得られた四元数である。ところが、(1)式のまではデータの前後関係を固めるような配慮がなく0 となるような元に対する処理もないためハッシュ衝突の可能性が高い。したがって、ハッシュ衝突を避けるため、乗算によってハッシュ値を算出する前に、ブロックの順序を固定化するような何らかの処理が必要となる。図2では、あるブロックとその前後、計3つの連続するブロックを利用し、各ブロックの要素から a_n, b_n, c_n, d_n を利用して四元数Q_nを計算することを示している。

別のデータから同じ四元数が生成可能であった場合、ハッシュ衝突が発生しやすい。この対策のため、QHFのアルゴリズムではあるブロックとその前後のブロックの要素を利用し、ブロックの前後関係を強固なものにしている。1つのブロック内でデータの攪拌を行うだけではハッシュ関数の堅牢性を高めるためには不十分である。このため、計算中に異なるブロックから要素を利用することが必要である。このQHFには図2に示すように4段階の前処理がある。これらについて以下に示す。



第 1 段階：q_{n-1}、q_n、q_{n+1} を生成する。
第 2 段階：q_{n-1}、q_n、q_{n+1} を利用し、XOR の計算をして X_1 から X_4 を求める。
第 3 段階：X_1 から X_4 を利用し、左リングシフトの計算をして Y_1Y_2 を求める。
第 4 段階：$Q_n=Y_1Y_2\overline{Y_1}$ を計算し、Q_n を求める。

図 2　QHF の前処理

ⓒ 2010 Information Processing Society of Japan

この前処理ではただ四元数の演算を行うだけではなくビットシフトも行っている。このビットシフトはs_nを利用したリングシフトである。Mには巨大素数、もしくはそれらの積を利用する。高い衝突耐性を得るために、Mは128ビット以上の数である必要がある。前処理においてQ_nを生成しているが、計算中に要素の値がすべて0であるようなブロックが存在する可能性がある。入力データから得たブロックで値が0となるものが1つでも存在すれば演算Qで対策される。しかし、それ以上の数の連続ブロックが0であるような場合にはQ_nが0となり、以降の計算もすべて0となってしまう。そこで、0となるブロックの要素数mを数え上げ、要素が0となるブロックの直前のブロックQ_nの値を利用した関数を利用する。この式で得られる四元数を0の連続部分のブロックの要素として利用する。

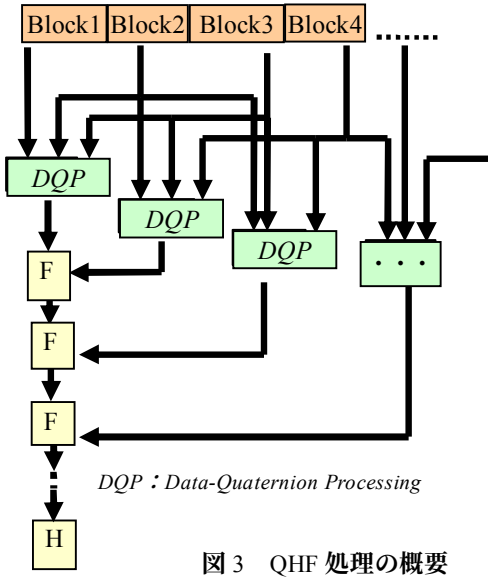


図 3　QHF 処理の概要

以上で、四元数をベースとし、法 M を利用するハッシュ関数 QHF の処理を示した。この関数の出力する数値は $Q = (w, x, y, z)$と M の 5 つの数で表現する。その形式は w-x-y-z-M のようになり、具体的には表１のような結果が出力される。表１の２種のファイルを比較すると、サイズが同じでファイル形式が違うが、ハッシュ値はまったく別の数値を出力しており、また近しい数値でもない。ハッシュ値生成は成功と言える。ただし、処理時間が長いことが表１より理解できる。

表 1　QHF の出力結果

ファイル形式	**JPG ファイル**
ファイルサイズ	18713 バイト
ハッシュ値	2BFFA693AFA67223-2D4C51B7AF915B19 - E111472C55D2978 - 3F121617DB96240F - 513F84F069ABE83F
処理時間	10.375 秒
ファイル形式	**text ファイル**
ファイルサイズ	18713 バイト
ハッシュ値	16F2A12D3721ACAA -21EB19A7B2A3D2A5 - 4F71FDF973D2B44F - 47C4EC5C7A365F94 - 513F84F069ABE83F
処理時間	10.406 秒

表 2　データ改竄における耐性の検証比較

変更点	ハッシュ値
オリジナル	3D9A76CEDB419BA1 - 3D8D53A30888C63C - 293A4724282BBE13 - 4B42B32D27B74B00 - 513F84F069ABE83F
1 バイト追加	3F9C4A533039A9AF - 27D090DED1C81A59 - 39603278A4B0F9A7 - 35150DD7457B77AF - 513F84F069ABE83F
1 バイト削除	388A6E8BA18C8CB0 - 3065028DDCC57E7F - 20F9B2F834641166 - 4D798B03200E4B35 - 513F84F069ABE83F
1 バイト変更	2D07CE3E07FF75A6 - 2455CCEDD3E96F83 - 3C724316A66EE915 - 3637319A44DEDF86 - 513F84F069ABE83F

いずれの結果も、オリジナルファイルの結果とは完全に異なる数値が生成されており、近しい数値でもないことがこの表 2 から理解することができる。このことからデータ改竄における耐性は極めて高いと考えられる。また、一様乱数列の入力データから切り出したブロック列によって生成した四元数列で考えると、これは四元数の要素

ⓒ 2010 Information Processing Society of Japan

長 n によって決定し、一致する確率は $1/2^{3n}$ である。四元数が完全に一致して $p=q$ となる確率は $1/2^{4n}$ である。*QHF* の強度も要素長に依存すると考えられるため、テスト版のように要素長 n=128 であれば、ベクトル部分の一致確率 $1/2^{384}$ で、四元数の完全一致確率は $1/2^{512}$ である。このことから衝突の可能性は低いと考えられる。また、QHF と既存のハッシュ関数との比較を行った。実行例として、WMA ファイル（759845 バイト）を利用した出力結果を表 3 に示す。

表 3　QHF と既存のハッシュ関数のハッシュ値

ハッシュ関数	ハッシュ値
SHA-1	a99e603e0b6b8345de7c4542 99336ad5c5fce3dd
SHA-256	854067b61d713a707a9d8607 3ea9d590241336699b10bfafb 5411628cecbe34
MD5	f49958939a1300ca120b22f4e f4e213b
QHF	29711FA85A43D79E － 10B1F30836D1944A － 3F2AAE59EC6A3A50 － 45558996712769A3 － 513F84F069ABE83F

QHF の問題点は、処理速度の遅さという点であると考えられる。しかし、QHF は SHA-1、SHA-2、MD5 のいずれと比較しても長いハッシュ値を生成するというメリットがある。また、ハッシュ値のラストは剰余演算の法であるため、鍵つきハッシュ関数として利用できるという特徴もある。低速であることが問題ではあるが、逆に処理が熟成し高速化することができれば MD5 などよりも広い応用が考えられる。また、処理の性質上暗号へと発展させうる面があるため、将来性はあるといえる。

4. まとめ

本報告では、Quaternion（四元数）演算を基礎とするハッシュ関数 QHF を扱い、ハッシュ値生成や衝突耐性の向上などの比較を行った。より高度な衝突耐性を得るために可能性のある全ての攻撃について考察した。そして、同サイズの異なる形式でのハッシュ値の生成と速度比較を行った。さらに、text ファイルの改竄に対するシミュレーションも行った。この場合のハッシュ値の生成結果はデータ改竄に強いことを示した。QHF に対しての攻撃確率も評価し、要素長によってさらに低くすることも可能であることを示した。また、QHF では法 M が固定であるが、実際には M は乱数によって生成し、衝突しないことを保障するほどの長さを持つことが必要である。

今後は、速度と安全性を同時に向上させることである。3 シフト型 QHF でも速度減少率は 3～4 割であるが、さらなる改良を加える必要がある。

参考文献

1) R. L Rivest, “The MD5 Message Digest Algorithm,” Request for Comments (RFC)1321, Internet Activities Board, Internet PrivacZ Task Force, 3RIPEMD-1281, April 1992..
2) H. Dobbertin, Cryptanalysis of MD5 compress, presented at the rump session of EurocrZpt'96, 1996.
3) Xiaoyun Wang, Collisions for Hash Functions MD4, MD5, HAVAL-128 and RIPEMD,Crypto'04,E-print, 2004.
4) Xiaoyun Wang; Hongbo Yu, "How to Break MD5 and Other Hash Functions,” ISBN 3-540-25910-4, EUROCRYPT2005.
5) Yuval Ishai, Eyal Kushilevitz, Rafail Ostrovsky, “Sufficient Conditions for Collision-Risistant Hashing,” In Proceedings of Second Theory of Cryptography Conference (TCC) Springer-Verlag Lecture Notes in Computer Science, 2005
6) Lenore Blum, Manuel Blum, and Michael Shub. "Comparison of two pseudo-random number generators", *Advances in Cryptology: Proceedings of Crypto '82*.
7) T. Nagase; M. Komata, T. Araki, “Secure signals transmission based on quaternion encryption scheme,”AINA04, Volume 2, pp.35 – 38, 2004.

ⓒ 2010 Information Processing Society of Japan