科学研究費助成事業　　研究成果報告書

平成 ２６ 年　 ４ 月　 ３ 日現在

機関番号：１４１０１
研究種目：基盤研究(C)
研究期間：2011～2013
課題番号：２３５０００８６

研究課題名（和文）非再帰型擬似乱数生成アルゴリズムを応用したハッシュ関数の研究

研究課題名（英文）Research of hash functions based on algorithm of nonrecursive pseudorandom number generator

研究代表者
谷口　礼偉（YAGUCHI, HIROTAKE）

三重大学・教育学部・特任教授

研究者番号：４０１５７９７０

交付決定額（研究期間全体）：（直接経費）　　2,300,000円、（間接経費）　　690,000円

研究成果の概要（和文）：本研究では，最近導入された非再帰型擬似乱数生成アルゴリズムSSI32を発展させて，多倍長整数乗算と乗算結果のシフトを繰り返すことにより 160，192，・・・，4096 ビット長のハッシュ値を持つハッシュ関数 MBnhash を構成した。MBnhash のアルゴリズムは，数学的には [1,2) 上のベータ変換 M(t)=βt-[βt]+1 の繰り返しとしてとらえることができ，従来の代数的なハッシュ関数とは大変異なるものである。また，そのセキュリティが，「５次以上の代数方程式を解く一般的なアルゴリズムは存在しない」に基づいて保証されることを示した。

研究成果の概要（英文）：In this research, developing the algorithm of recently introduced nonrecursive pseudorandom number generator SSI32, we constructed, by repeating multiplication of integers and shift, the hash function MBnhash whose hash values are 160, 192, ..., 4092. Mathematically we can regard the algorithm of MBnhash as repetition of beta-transformation M(t)=bt-[bt]+1 (b=beta) on [1,2), which is quite different from other algebraic hash functions. We also showed the security of MBnhash based on the fact that we have no general algorithm of solving algebraic equations whose degrees are higher than 4.

研究分野：応用数学

科研費の分科・細目：情報学・計算機システム・ネットワーク

キーワード：ハッシュ関数　ネットワーク　セキュリティ　アルゴリズム　乱数　ベータ変換　代数方程式　超越数

１．研究開始当初の背景

(1) コンピュータの高性能化と大衆化により，従来のハッシュ関数が生成するビット長ではセキュリティが不十分であると考えられるようになってきた。そのため，長いビット長のハッシュ値を統一的なアルゴリムで生成するハッシュ関数が必要な状態になっていた。

(2) 一方で，カオス写像のエルゴード性を利用した非再帰型３２ビット擬似乱数生成法 SSI32 が新しく発表されていた。そのアルゴリズムは，従来の代数的な乱数生成法と異なり，「整数の乗算と乗算結果のシフト」の繰り返しで構成され，数学的には区間 [1,2) 上のベータ変換の繰り返しとしてとらえられるものであった。

(3) SSI32 のアルゴリズムは非常にシンプルで柔軟性に富むものであり，長いハッシュ値長をもつハッシュ関数への発展・応用に十分耐えられるものであった。

２．研究の目的

本研究の目標は２つある。

(1) １つは非再帰型乱数生成アルゴリズム SSI32 を発展させて，160, 192, 256, 384, 512, 1024, 2048, 4096 ビット長のハッシュ値を生成する新しいハッシュ関数 MB*n*hash を構成することである。新しいアルゴリズムで統計的に良い性質を持つハッシュ関数を構成すること自体が重要な意味を持っている。

(2) ２つ目は，新しく構成されたハッシュ関数 MB*n*hash の情報セキュリティ面での安全性を示すことである。ハッシュ関数は，情報伝達におけるメッセージ認証に使われることが多く，悪意ある第三者の攻撃に対する耐性を示す必要がある。そのため，MB*n*hash がハッシュ値を生成するアルゴリズムを数式化して，計算量の理論などと結びつけ，安全性を解析することである。

３．研究の方法

(1) ［ハッシュ関数の構成］長いハッシュ値を持つハッシュ関数 MB*n*hash の構成に当たっては，まず，３２ビットのハッシュ値をもつ小型ハッシュ関数 MB32hash を構成し，それが，様々な入力に対して十分な乱数値（ハッシュ値）を生成することを確認した。続いて，そのアルゴリズムを n=160, ・・・, 4096 ビットのハッシュ関数 MB*n*hash に拡張するという方法で研究を行った。

(2) ［ハッシュ関数の安全性］新しいハッシュ関数 MB*n*hash は，入力バイト列の圧縮と，圧縮結果の攪乱という２つのプロセスで構成されているので，それぞれのプロセスについて，

①まず，アルゴリズムそのものの安全性を証明する；

②続いて，アルゴリズムを計算機に実装した際に生じる安全性の減少・欠落を解析する；

という方法で研究を行った。

４．研究成果

新しいハッシュ関数は，

・入力バイト列の圧縮（compression），

・圧縮結果の攪乱（scrambling）

という２つのプロセスから構成され，何れも [1,2) 上のベータ変換

$$M_\beta(t) = \beta t \mod [1,2) = \beta t - \lfloor \beta t \rfloor + 1$$

を使っている。（$\lfloor x \rfloor$ は x の整数部分）

(1) ［小型ハッシュ関数 MB32hash の構築］

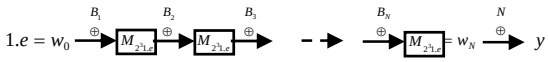
① MB32hash のアルゴリズムを構築する。

［圧縮過程］$1.e = 1.27181\cdots$ とし，入力バイト列を $\mathsf{B} = B_1 B_2 \cdots B_N$ とする。

$w_0 = 1.e$ とし，w_{k-1} から w_k を

$$w_k = M_{2^3 1.e}(w_{k-1} \oplus B_k), \quad k = 1,2,\cdots,N$$

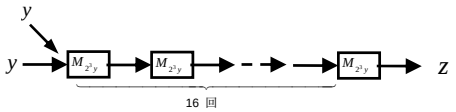
により計算し，得られた w_N にデータ長 N を埋め込んで $y \equiv w_N \oplus N$ を得る。



［攪乱過程］非再帰型３２ビット擬似乱数生成法 SSI32 のアルゴリズムを応用して，圧縮過程で得られた y に対応した３２ビット乱数値（=ハッシュ値） を

$$z = (M_{2^3 y})^{16}(y) \qquad \zeta = \lfloor 2^{32}\left(2^{11} z - \lfloor 2^{11} z \rfloor\right) \rfloor$$

により取り出す。



② ①で構築されたアルゴリズムを計算機に実装する。①の MB32hash アルゴリズムの記述では，計算機が使う各変数のビットサイズは考慮されていない（実質無限長）。実際の計算機では有限サイズであるので，以下の３２ビット系で計算機に実装する。

・[1,2) 内のすべての実数は，（乗算結果を除いて）３２ビット $1.b_1 b_2 b_3 \cdots b_{31}$ で表される（３２ビットを超す部分は切り捨てる），

・２数 $x = 1.x_1 x_2 \cdots x_{31}$ と $t = 1.t_1 t_2 \cdots t_{31}$ の乗算結果は６４ビット $\tilde{b}_0 \tilde{b}_1 . b_2 b_3 \cdots b_{63}$ で表され，その後，左シフト等を経て３２ビットに切り詰められる。

この３２ビット実数系は，整数３２ビット系と同一視できるので，$M_{2^3 x}(t)$ の実際の計算は

$$1x_1x_2\cdots x_{31} \xrightarrow{\text{整数乗算}} \tilde{b}_0\tilde{b}_1\tilde{b}_2\tilde{b}_3\tilde{b}_4\tilde{b}_5\cdots\tilde{b}_{35}\cdots\tilde{b}_{63}$$
$$1t_1t_2\cdots t_{31}$$

$$\xrightarrow{\text{左シフト4}} \tilde{b}_4\tilde{b}_5\cdots\tilde{b}_{35}\cdots\tilde{b}_{63} \xrightarrow{\text{OR}}$$

$$1\tilde{b}_5\cdots\tilde{b}_{35}\cdots\tilde{b}_{63} \xrightarrow{\text{切り詰め}} 1\tilde{b}_5\cdots\tilde{b}_{35}$$

のように簡単な整数演算で実現できる。

③ ②で実装されたハッシュ関数が生成するハッシュ値の乱数性を検証する。ハッシュ関数は入力バイト列に対応した乱数を出力する乱数生成器と考えられる。したがってその出力（＝ハッシュ値）は十分な乱数性を有さなければならない。実装された MB32hash の乱数性を NIST の検定により検証した。188 のテストから構成される NIST の検定は計１０回行われ，各テストが出力する p 値および検定をパスした比率について，その平均をテスト毎にとり，さらに各平均値の最大・最小を求めた。結果は以下の通りである：

P-値　　AvMax = 0.737001　AvMin = 0.285480

パス比率 AvMax = 0.992503　AvMin = 0.986800

良く知られている乱数生成器 Mersenne Twister ar についての結果は

P-値　　AvMax = 0.713578　AvMin = 0.188745

パス比率 AvMax = 0.992879　AvMin = 0.986809

であるので，MB32hash の乱数性には特に問題が無いことが分かる。

(2)［ハッシュ関数 MB$n$hash の構築］(n=160, 192, ..., 4096) MB32hash のアルゴリズムは単純で柔軟性に富んでいるので，ハッシュ値のサイズを自由に設定することが可能である。具体的には圧縮過程では $M_{2^3 1.e}$ の代わりに $M_{2^{p-1}1.e}$ を使い，撹乱過程では $M_{2^3 y}$ の代わりに $M_{2^{S-1}y}$ を使う。また，圧縮過程では１度に埋め込むバイト数を１バイトから E バイトに増やす。n ビットのハッシュ値 ς は

$$\mathsf{Z} = (M_{2^{S-1}\hat{\mathsf{W}}})^{16}(\tilde{\mathsf{W}}) \quad \zeta = \left\lfloor 2^n (2^{(Q/2)-S-1}\mathsf{Z} - \left\lfloor 2^{(Q/2)-S-1}\mathsf{Z} \right\rfloor) \right\rfloor$$

により取り出す。

(3)［ハッシュ関数 MB$n$hash の実装］計算機への実装に当たっては，$M_x(t)$ の x および t に n ビットを割り当てると乗算のスピードが遅くなるので，x に割り当てるビットを節約して，圧縮過程では M ビット，撹乱過程では Q ビットを割り当てる。MB32hash での各値（バイト）は次の通りである：

MB$n$hash	M (size of $1.e$)	E (bytes Embeded.)	P	Q (size of $\hat{W}$)	S
160	8	8	5	12	2
192	8	8	5	12	2
256	8	8	5	12	3
384	12	16	9	12	4
512	12	16	9	16	6
1024	20	32	17	28	12
2048	36	64	33	52	24
4096	68	128	65	96	48

(4)［MB$n$hash の乱数性］MB$n$hash の生成するハッシュ値の乱数性を TestU01 の Crush テストにより検証した。TestU01 は NIST の検定より強力で，Crush テストでは検定の過程において約 3×10^{10} 個の３２ビット整数もしくは [0,1] 内に値を持つ倍精度実数を消費する。MB$n$hash の各 n について，生成されるハッシュ値の全ビットおよび最初の３２ビットに対してそれぞれ Crush テストを実施した。結果はもちろん All tests were passed である。最も時間がかかった検定は当然 MB4096hash の最初の３２ビットの場合であり，約 6600 時間であった（並列計算）。

(5)［MB$n$hash のスピード］MB$n$hash がハッシュ値を生成する速度は以下の通りである。（OS: Windows 7 Professional 64-bit, CPU: Xeon 3.1GHz, メモリー: 8GB, C コンパイラー: Intel icl v.12）

① 入力バイト列が非常に長いとき（1024^3 バイト）に，対応するハッシュ値を生成するのに要する時間：

Mbnhash →	160	192	256	384	512
time(32-bit)	6.3	7	8.3	7.7	9.7
time(64-bit)	-	5.3	6	5.7	6.3

1024	2048	4096	(SHA512)(1)	(SHA512)(2)
14.7	22.3	37	16.7	5.7
8.3	11.3	17.7	-	-

表中で，(SHA512)(1), (SHA512)(2) は SHA512 の異なるソースコードである。また，(32-bit), (64-bit) は，MB$n$hash の計算中に現れる多倍長の乗算が，それぞれ (32-bit)×(32-bit)＝(64-bit) および (64-bit)×(64-bit)＝(128-bit) で行われたことを表している。

②入力バイト列そのものは非常に短い（８バイト）が，多数（10^7）の入力データが存在する場合に，全てのハッシュ値を生成するのに要する時間：

Mbnhash →	160	192	256	384	512
time(32-bit)	6.7	8	9.3	13	22.7
time(64-bit)	-	5.3	7	10.3	11.3

1024	2048	4096	(SHA512)(1)	(SHA512)(2)
80.3	274.7	965.7	26	6.7
40.7	128.7	400	-	-

(6)［アルゴリズムの観点から見た MB$n$hash の安全性］まず，アルゴリズムがどのように実装されるかを考慮しないで，アルゴリズムそのものの安全性を考える（＝各実数は無限のビット長で表されていると仮定する）。また，安全性の解析は n によらないので，$n=32$ として考える。

① ［圧縮過程のアルゴリズムの安全性］２つの入力バイト列 $\mathsf{B} = B_1B_2\cdots B_N$ と $\hat{\mathsf{B}} = \hat{B}_1\hat{B}_2\cdots\hat{B}_{\hat{N}}$ に対する圧縮後の値 w_N と $\hat{w}_{\hat{N}}$ が，どのような時に $w_N = \hat{w}_{\hat{N}}$ となるかを調べる。$\gamma \equiv 2^3(1.e)$ とおく。e は超越数（有理係数の代数方程式の解とならない数）であることに留意する。$w_{k-1}\oplus B_k = w_{k-1}\pm t_k$（$t_k$ は

８ビット）と表せば，

$$w_N = \gamma^N(1.e+t_1)+\gamma^{N-1}(\tilde{p}_1+t_2)+\cdots+\gamma(\tilde{p}_{N-1}+t_N)+\tilde{p}_N$$

となる。ただし，$\tilde{p}_k \in \{-19,-18,\cdots,-9\}$である。同様に，$\tilde{q}_k \in \{-19,-18,\cdots,-9\}$ として，

$$\hat{w}_{\hat{N}} = \gamma^{\hat{N}}(1.e+\hat{t}_1)+\gamma^{\hat{N}-1}(\tilde{q}_1+\hat{t}_2)+\cdots+\gamma(\tilde{q}_{\hat{N}-1}+\hat{t}_{\hat{N}})+\tilde{q}_{\hat{N}}$$

となる。従って $w_N = \hat{w}_{\hat{N}}$ とすれば，γ は代数方程式の解でなければならない。しかしながら，$\tilde{p}_k, \tilde{q}_k, t_k, \hat{t}_k$に関する条件から許される代数方程式の数は高々 $(2^9)^{N+\hat{N}}\times 11^{N+\hat{N}}$ であり，Ｂ≠Ｂ̂ のときこのような数の代数方程式の解の中に γ が含まれることは不可能である。従って $w_N = \hat{w}_{\hat{N}}$ となるのは，Ｂ＝Ｂ̂ の場合に限られ，圧縮過程はアルゴリズムとして安全であることが分かる。

② ［撹乱過程のアルゴリズムの安全性］

２つの入力バイト列を圧縮後に得られる

$$y \equiv w_N \oplus N \qquad \hat{y} \equiv \hat{w}_{\hat{N}} \oplus \hat{N}$$

がどのような時に同じハッシュ値 $\varsigma = \hat{\varsigma}$ を生成するかを調べる。

$$z = (M_{2^3 y})^{16}(y) \qquad \zeta = \lfloor 2^{32}\left(2^{11}z - \lfloor 2^{11}z \rfloor\right)\rfloor$$

であるので，

$$z(y) = (2^3 y)^{16} y + (2^3 y)^{15}\tilde{p}_1 + \cdots + (2^3 y)\tilde{p}_{15} + \tilde{p}_{16}$$
$$\hat{z}(\hat{y}) = (2^3 \hat{y})^{16}\hat{y} + (2^3 \hat{y})^{15}\tilde{q}_1 + \cdots + (2^3 \hat{y})\tilde{q}_{15} + \tilde{q}_{16}$$

と表される，ただし $\tilde{p}_k, \tilde{q}_k \in \{-7,-8,\cdots,-30\}$ である。$\varsigma = \hat{\varsigma}$ であるためには

$$(\#) \quad \left|\left(z - 2^{-11}\lfloor 2^{11}z\rfloor\right) - \left(\hat{z} - 2^{-11}\lfloor 2^{11}\hat{z}\rfloor\right)\right| < 2^{-43}$$

が必要であるので，$|y-\hat{y}| > 2^{-91}$ であって，$\varsigma = \hat{\varsigma}$ となるような $y, \hat{y}$ を求めようとすると，上記の $y, \hat{y}$ に関する代数不等式を解かなければならない。しかしながら，５次以上の代数方程式を解く一般的なアルゴリズムは無いので，不等式を解くことは非常に難しい。$\hat{\zeta}$ は任意に固定されたものであるので，どのような $\hat{\zeta}$ についても不等式を解くことが困難となり，従って上記不等式を理論的に解くと言うことは大変困難であることが分かる。これから，$\varsigma = \hat{\varsigma}$ となるような $y, \hat{y}$ を求めることは非常に困難であることが分かる。また例え $y, \hat{y}$ が見つかったとしても，圧縮後にそのような $y, \hat{y}$ を生成するような入力バイト列 Ｂ，Ｂ̂ を見つけることは前述の議論から甚だ困難であることが分かる。

(7) ［アルゴリズムの計算機への実装という観点から見た MB*n*hash の安全性］

①［圧縮過程の実装の安全性］MB32 hash アルゴリズムを３２ビット系 $1.b_1b_2b_3\cdots b_{31}$ で実装すると，各数は最初の３２ビットのみで表されそれ以下のビットは棄てられる。したがって，$w_{k-1}\oplus B_k$ の $\oplus B_k$ を w_{k-1} の LSB（最小位ビット）b_{31} を含む $b_{24}\cdots b_{31}$ に作用させるようにしてしまうと，異なる入力バイト列 Ｂ，Ｂ̂ で同じハッシュ値 $\varsigma, \hat{\varsigma}$ を生成させることが出来てしまう。これを避けるには，$\oplus B_k$ で少なくとも LSB に触らせない（避ける）ようにしておけば良い。また，一般に，$M_{2^{p-1}1.e}$ 中の $1.e$ の n ビット化においては，LSB=1 とすると，乱数性の向上および安全性の向上が図られる。（撹乱過程の $M_{2^{s-1}y}$ においても同様である）

②［撹乱過程の実装の安全性］(6)の②に現れる不等式 (#) を理論的に解くことは難しくても，数値計算で解を求めておいて，それを参考にして３２ビットの解を求めることが出来るのではないかと思われる。しかしながら，y と $y+\eta$ の生成するハッシュ値を $\varsigma, \hat{\varsigma}$ とするとき，$\eta > 2^{-91}$ ならば $\varsigma \neq \hat{\varsigma}$ となるので，数値解を３２ビットに丸めた時点でハッシュ値が変わってしまい，数値解は役に立たないことが分かる。

(8)［M_β のエルゴード的性質］M_β は [0,1) 上の linear mod 1 変換 $T_{\beta,\hat{\beta}}$ と同型

$$M_\beta(t) = T_{\beta,\hat{\beta}}(t-1)+1, \quad t\in[1,2)$$

であるので（$\hat{\beta}$ は β の小数部分），エルゴード理論の様々な結果が使える。特に，不変測度を与える密度関数 $h_{\beta,\hat{\beta}}$ について

$$1-\frac{1}{\beta-1} \le h_{\beta,\hat{\beta}}(x) \le 1+\frac{1}{\beta-1}$$

の評価が得られ，例えば $n=1024$ の時には，$2^{95} \le \beta < 2^{96}$ などとなり，β, x の値によらずほぼ１に近い値を取ることが分かる。

(9)［SSI64rand の構成］最近は殆どの計算機で 64 ビットの演算が可能になっている。一方 SSI32rand の構成はアルゴリズムの記述先行で，きちんとした数式による記述がなかった。そこで将来に備えて， SSI64rand のアルゴリズムを[1,2)上のベータ変換の繰り返しとして数式的にきちんと記述し，さらに 64 ビットの乗算とシフトにより SSI64rand を構成し，乱数性を検証した。

(10)［研究の位置づけ，インパクト，展望］本研究は，従来とは全く異なる方法でハッシュ関数を構成し，その安全性を示したものであるから，この分野でのインパクトはかなり高いと考えられる。しかしながら，手法が従来の代数学的な方法とは異なる解析学的なものであり学際的なものであるため，理解されるまでにかなりの時間を要すると思われる。例えば，本研究の結果をまとめた投稿論文 Construction and security of non-algebraic hash functions based on β-transformations on [1,2) の査読が，１年以上経った現在でも終わっていない。本研究を少しでも理解して頂くため，研究の詳細を「研究報告資料」としてまとめ，当報告者のホームページに掲載した。

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計２件）

① Yaguchi, H., Construction of a nonrecursive 64-bit pseudorandom number generator based

on beta transformations on [1,2). Bulletin of the Faculty of Education, Mie University（三重大学教育学部研究紀要）, 査読無 , 65 19-25 (2014). http://miuse.mie-u.ac.jp/

② Yaguchi, H. and Ueda, S., Construction, randomness and security of new hash functions derived from chaos mappings. Interdisciplinary Information Sciences, 査読有 , 18 No. 1, 1-11 (2012). DOI 10.4036/iis.2012.1

〔学会発表〕（計３件）

① Yaguchi, H., Construction and security of hash functions based on β-transformations on [1,2). Ergodic Theory and Metric Number Theory, 2012年12月6日, 日本女子大学 目白キャンパス

② 谷口礼偉, [1,2) 上のβ変換を用いたハッシュ関数の構成と安全性. 日本数学会2012年度秋季総合分科会, 2012年9月19日, 九州大学 伊都キャンパス

③ Yaguchi, H., Construction and security of a non-algebraic tiny and extensible hash function. ASIACRYPT 2011 Rump Session, 2011年12月6日, 韓国ソウル市ソウル新羅（シーラ）ホテル

〔その他〕
ホームページ等
http://math1.edu.mie-u.ac.jp/yaguchi/kakenDocument_2013.pdf（当研究成果報告書の補助資料）
http://math1.edu.mie-u.ac.jp/yaguchi/MBnhash5BnoMain.c（MB*n*hash（*n*=160, 192, ..., 4096）のソースコード）
http://math1.edu.mie-u.ac.jp/yaguchi/SSI64rand.c（SSI64randのソースコード）

６．研究組織

(1)研究代表者
谷口　礼偉（YAGUCHI, Hirotake）
三重大学・教育学部・特任教授
研究者番号：40157970

(2)研究分担者
（　　）
研究者番号：

(3)連携研究者
（　　）
研究者番号：