GPGPUによるレインボーテーブル生成の高速化

兼松 卓也[1,a)]　桑原 寛明[2]　上原 哲太郎[3]　國枝 義敏[3]

概要：レインボーテーブルはパスワード解析に利用されるパスワードとハッシュ値の対応表である．レインボーテーブルはパスワードとハッシュ値の組を計算によって復元可能とすることでコンパクトな表となっているが，生成には膨大な計算時間を要する．本稿では，一般的なレインボーテーブルよりも効率的なパスワード解析が可能なレインボーテーブルを GPGPU を用いて高速に生成する手法を提案する．数字，英小文字，英大文字で構成される 4 文字のパスワードに対応するレインボーテーブルを提案手法によって生成し，GPGPU による高速化の効果を示す．生成したレインボーテーブルを用いてパスワード解析を行い，効率化による解析時間の変化を示す．

キーワード：パスワード，レインボーテーブル，並列処理，GPGPU

Acceleration of Generating Rainbow Tables by GPGPU

Takuya Kanematsu[1,a)]　Hiroaki Kuwabara[2]　Tetsutaro Uehara[3]　Yoshitoshi Kunieda[3]

Abstract: A rainbow tables is a map of password and hash value to be used for password cracking. This is a compact table by compressing passwords and hash values, but rainbow tables generation takes an enormous amount of computing time. We propose a method to accelerate rainbow tables generation using GPGPU. Generated rainbow tables can achieve more efficient password cracking than ordinary rainbow tables. We apply our proposed method to generate rainbow tables containing passwords composed of 4 lowercase and uppercase letters and numbers for showing the effect of speeding up ongeneration, and crack password by generated rainbow tables.

Keywords: Password, Rainbow tables, Parallel processing, GPGPU

1. はじめに

情報社会において一般的に用いられる個人認証技術の 1 つにパスワードシステムがある．パスワードシステムではパスワードと ID(identification:各個人で異なる識別子) が用いられる．認証時に入力された ID とパスワードの組み合わせが事前に登録されたものと一致すれば，入力者が ID 登録者本人であるとみなす．パスワードは他者に知られるとなりすましをされる危険性があるため，流出に備える必要がある．そこで一般的なパスワードシステムでは，登録したパスワードはハッシュ関数によってハッシュ値に変換して保存される．認証時には入力されたパスワードをハッシュ値に変換し，事前に登録されているハッシュ値と一致判定を行う．

パスワード解析とはハッシュ値から元のパスワードを導き出すことである．代表的なパスワード解析手法に総当たり攻撃と辞書攻撃がある [1]．総当たり攻撃は攻撃対象のパスワードシステムで使用可能なすべてのパスワードを候補として試行することによって正しいパスワードを探索する手法である．総当たり攻撃はパスワードの文字種，文字数が増えるとパスワード候補が増加し，膨大な計算時間を

[1] 立命館大学大学院 情報理工学研究科
Graduate School of Information Science and Engineering, Ritsumeikan University
[2] 南山大学 情報センター
Center for Information and Communication Technology, Nanzan University
[3] 立命館大学 情報理工学部
Department of Computer Science, Ritsumeikan University
[a)] is0109iv@ed.ritsumei.ac.jp

必要とする．辞書攻撃はパスワードに使用されやすいとされる文字列をパスワードを候補として試行することによって正しいパスワードを探索する手法である．辞書攻撃はパスワードの文字種，文字数が増えると辞書に登録するパスワード候補が増加し，膨大な記憶容量を必要とする．これらの手法への対策として，パスワードの文字数を増やす，パスワードに使われやすい文字列をパスワードに使用しないといった方法がある．この対策は容易に行えるため総当たり攻撃，辞書攻撃は計算量，記憶容量的観点からパスワード認証の現実的な脅威にはなりにくいとされている．

空間的，計算量的な問題を解決するパスワード解析手法にレインボークラックがある [2]．レインボークラックはレインボーテーブルを用いることによって，現実的な記憶容量や計算量でのパスワード解析を可能とする．レインボーテーブルは大量のパスワードとそのハッシュ値のペアを圧縮して保存することによって，必要な記憶容量を削減したデータテーブルである．レインボーテーブルは一度生成すれば，辞書攻撃に用いられる辞書と同様に再利用可能である．しかし，レインボーテーブルの生成では総当たり攻撃と同程度の計算処理とデータの圧縮処理を行うため，膨大な時間が必要である [3]．よって，現代のパスワードシステムではレインボークラックは大きな脅威ではないとされている．

本研究では広く普及しているアーキテクチャを使用したレインボーテーブルの高速な生成手法を提案することによって，レインボークラックによるパスワード解析の脅威がどの程度現実的となっているかを示す．提案手法では，レインボーテーブルの生成に大量のデータを並列処理可能な GPU(Graphics Processing Unit) を GPGPU(General Purpose computing on GPU) として用いる．GPU を用いてレインボーテーブルの生成処理を並列化し，高速なレインボーテーブルの生成を実現する．並列化したレインボーテーブル生成処理では，投機的にパスワードの生成を行うためパスワードの衝突が発生する．よって，提案手法ではパスワードの重複を含むレインボーテーブルが生成される．生成したレインボーテーブルはレインボークラックの効率化のために複数のファイルに分割して記録される．

評価実験では逐次処理および提案手法を用いた並列処理によるレインボーテーブルの生成時間を比較し，GPU を用いることによる高速化の効果を示す．生成したレインボーテーブルの性能はパスワードの網羅率を算出することによって明らかにする．生成したレインボーテーブルを用いたレインボークラックの所要時間を計測し，テーブル分割によるパスワード解析の効率化の効果を示す．

2. レインボークラック

レインボークラックは与えられたハッシュ値に対応するパスワード (以降，平文と表記する) をレインボーテーブルを用いて導出するパスワード解析手法である．

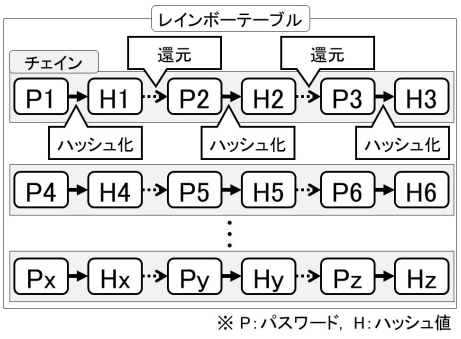

図 1 レインボーテーブルの構造
Fig. 1 Structure of rainbow tables

2.1 レインボーテーブル

レインボーテーブルはチェインの集まりからなるデータテーブルである．チェインは，チェイン化された平文とハッシュ値のペアの集まりである．チェイン化とはハッシュ値から適当な平文を生成し，平文とそのハッシュ値のペアを他のペアと関連付けることである．レインボーテーブルの生成では，ハッシュ値の生成にハッシュ関数を使用し，チェイン化に還元関数を使用する．テーブルの生成に用いたハッシュ関数と還元関数はレインボークラックにも用いる．

レインボーテーブルの構造を図 1 に示す．図 1 はチェインの長さが 3 であるレインボーテーブルの構造である．ここで，P は平文，H はハッシュ値を表す．平文と対になるハッシュ値はハッシュ関数を用いたハッシュ化によって生成される．この平文とハッシュ値のペアは還元関数による還元によって他のペアとチェイン化される．

2.1.1 ハッシュ関数

暗号学的ハッシュ関数 (以降，ハッシュ関数と呼ぶ) は，入力された任意長のビット列をもとに，規則性のない固定長のビット列を生成する関数である [4]．ハッシュ関数によって生成されたビット列をハッシュ値と呼ぶ．ハッシュ関数を使用して平文からハッシュ値を生成することをハッシュ化と呼ぶ．

ハッシュ値の生成に用いられた平文がハッシュ値から逆算不可能である性質を一方向性と呼ぶ．一方向性が保証されているハッシュ関数を用いて生成されたハッシュ値は復号不可能である．

2.1.2 還元関数

還元関数は，入力されたハッシュ値から固定長の文字列を生成する関数である．還元関数によって文字列を生成することを還元と呼ぶ．還元関数によって生成された文字列を平文とするペアをチェイン化の対象とする．よって，還元で生成される文字列 (以降，混乱しない限り平文と呼ぶ) は生成するレインボーテーブルに含まれるべき平文の集合

に含まれている必要がある．チェインの再現性を保つために，還元で生成される平文は実行環境に依存しない必要がある．

異なるハッシュ値からある同一の平文が還元されることを平文の衝突と呼ぶ．平文の衝突は 2 種類に分けられる．1 つは異なるチェインにおいて同一の平文が還元されることによる平文の衝突，もう 1 つはある平文が 1 つのチェイン内で還元により複数回出現することによる平文の衝突である．レインボーテーブルの生成において，チェイン間で平文の衝突が発生すると，衝突した平文以降のチェインはすべて重複する．一方，1 つのチェイン内において平文が衝突すると繰返しが発生する．すなわち，この場合も衝突した平文以降に生成される平文は，そのチェイン内で既に生成済みの平文とすべて重複する．平文の衝突によるチェインの重複を避けるためには還元関数を複数用いるといった工夫が必要となる．衝突を完全に防ぐにはもととなるハッシュ値と還元される平文を 1 対 1 関係で結びつける必要があるが，そのような還元関数を作るのは困難である．

2.1.3 チェインの圧縮

レインボーテーブルは保存するチェインを圧縮することによって，レインボーテーブルを記録するための記憶資源の使用量を削減する．圧縮前後のチェインの構造を図 2 に示す．圧縮前のチェインの先頭の平文 (Start Point：以降，SP と表記する，図 2 中 P1) と末尾のハッシュ値 (End Point：以降，EP と表記する，図 2 中 H3) の 2 つの要素のみを取り出して記憶することでチェインを圧縮する．圧縮したチェインは，チェイン生成時に用いたハッシュ関数と還元関数を SP に対して交互に適用することで復元できる．復元の過程で EP と同一のハッシュ値が生成されたら復元完了とする．

図 1 のレインボーテーブルのチェインを圧縮したレインボーテーブルを図 3 に示す．チェインを圧縮した場合，レインボーテーブルはチェインの本数分の SP と EP のみで構成される．パスワード解析時の記憶資源の使用量を節約するため，レインボークラックにはチェインを圧縮したレインボーテーブルを用いる．以降，本稿では図 3 に示したような圧縮したレインボーテーブルを単にレインボーテーブルと呼ぶ．

2.1.4 レインボークラックの手順

レインボークラックにはレインボーテーブル，還元関数，ハッシュ関数を用いる．圧縮したチェインは，チェインの生成に用いた還元関数とハッシュ関数によってのみ復元できる．よって還元関数とハッシュ関数はレインボーテーブルの生成に使用した関数と同一である必要がある．

レインボークラックでは解析対象のハッシュ値 (Target Hash：以降，TH と表記する) を先頭とするチェインを生成する．このとき，ハッシュ値を生成する毎にレインボーテーブル上のすべての EP と生成したハッシュ値を比較する．生成したハッシュ値と同一の EP をもつチェインは，チェイン内に TH を含む．TH を含むチェインの SP を取得し，SP をもとにチェインを復元することによって TH に対応する平文が取得できる．

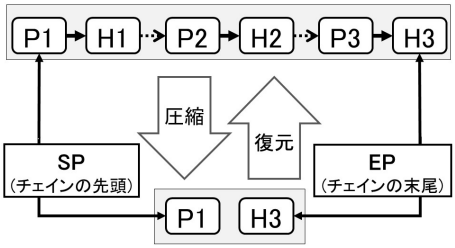

図 2 チェインの圧縮
Fig. 2 Compression of a chain

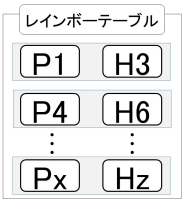

図 3 圧縮後のレインボーテーブル
Fig. 3 A compressed rainbow tables

レインボークラックの例を図 4 に示す．図 4 では P は平文，H はハッシュ値，破線の矢印は還元，実線の矢印はハッシュ化を表す．ここでは TH は H4 とする．まず，図 4 中 (1) において，H4 と一致する EP をレインボーテーブル内で探索する．一致する EP がないため，H4 を先頭にしたチェインを生成する．図 4 中 (2) において，生成した H5 と一致する EP が存在しないため，チェイン生成を続行する．図 4 中 (3) において，生成した H6 がレインボーテーブル上に EP として存在している．よって，テーブル上で H6 に対応した SP である P4 を取得する．図 4 中 (4) において，取得した P4 をもとにチェインの復元を開始する．図 4 中 (5) において，復元中のチェイン上に H4 が出現したためチェインの復元を中断する．復元したチェインから H4 に対応する平文は P4 であることが判明し，レインボークラック完了となる．

3. 提案手法

本研究ではレインボーテーブルの生成におけるチェイン生成処理を高速化するために，GPU と CPU で並列してチェインを生成する手法を提案する．また，レインボークラックにおける EP の検索を効率化するためにレインボーテーブルを分割して保存する手法を提案する．

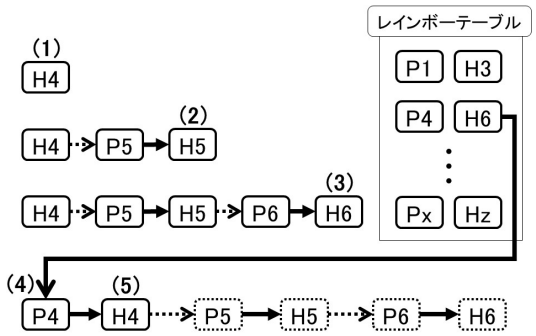

図 4　レインボークラックの例
Fig. 4　A example of rainbow crack

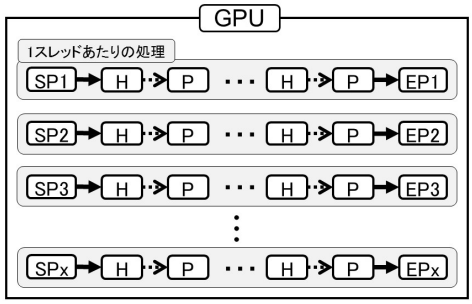

図 5　GPU でのチェイン生成処理
Fig. 5　Generating chains by GPU

3.1　チェイン生成処理

同一の SP からは同一のチェインが生成される．よって全ての SP は他の SP と重複しないように決定する．本提案手法では決定した SP をもとに平文の衝突を検査せずに一定長のチェインを生成するため，レインボーテーブル内に重複する平文が存在する可能性がある．未生成の平文の検出も行わないため，生成されたレインボーテーブルが網羅していない平文が存在する可能性がある．よって，平文の網羅率を向上させるためには十分な数のチェインを生成する必要がある．

チェイン生成処理の終了条件としてチェインの長さを事前に決定する．生成中のチェインの長さが事前に決められた長さに到達したらチェイン生成処理を終了し，次のチェインの生成処理を開始する．よってすべてのチェインの長さは等しくなる．

レインボーテーブル生成処理の終了条件としてチェインの総数を事前に決定する．生成したチェインの総数が事前に決められた数に到達したらレインボーテーブルの生成処理を終了する．

3.2　平文の位置番号を利用する還元関数

チェインにおける平文の位置番号をハッシュ値の還元に用いる．平文の位置番号は還元される平文が SP から数えて何個目の平文にあたるかを表す．平文の位置番号を還元に用いることによって，生成される平文の位置番号が異なれば同じハッシュ値から異なる平文が還元される．チェイン内で衝突が発生しても，1 つのチェインにおいて平文の位置番号が重複することはないため，衝突した平文の次に還元される平文は衝突せず，チェインの繰り返しの発生を回避できる．チェイン間で平文が衝突しても，平文の位置番号が異なれば衝突した平文の次に生成される平文は衝突しないため，チェインの重複を回避できる．しかし異なるチェイン上において，平文の位置番号が同一である位置で平文の衝突が発生した場合，衝突した平文以降のチェインは重複し，EP の重複が発生する．EP の重複が発生してもチェインの内容は同一ではない．

レインボークラックでは TH と一致する EP が無ければ TH からチェインを生成し，新たに生成したハッシュ値と一致する EP を探索する．EP の重複によって，生成したハッシュ値と一致する EP を持つチェインが存在しても，復元したチェイン上に TH が存在しない可能性がある．よって，生成したハッシュ値と一致する EP を持つチェインが複数存在すれば，それらのチェインをすべて復元して TH の有無を調べる必要がある．

3.3　チェイン生成処理の並列化

本研究ではチェイン生成処理を CPU と GPU で並列して行うことによってレインボーテーブルの生成を高速化する．

3.3.1　GPU でのチェイン生成処理

GPU を汎用計算に利用するために CUDA(Compute Unified Device Architecture)[5] を用いる．CUDA における並列処理の最小処理単位をスレッドと呼ぶ．GPU でのチェイン生成処理を図 5 に示す．ここで P は平文，H はハッシュ値を表す．GPU を用いて並列化したチェイン生成処理では 1 つのスレッド上で 1 つのチェインを生成する．このとき，スレッド並列数は GPU のメモリ容量に制限される．GPU の 1 回の起動で 1 度に生成できるチェインの本数が事前に決定した生成したいチェインの総数に満たない場合，SP を変えながら GPU を複数回起動してチェイン生成処理を行う．本稿では GPU でチェイン生成処理を 1 回行うことを 1 ラウンドと表す．

3.3.2　CPU でのチェイン生成処理

GPU がチェイン生成処理を行っている間，CPU では以下の処理を順に行う．

(1) 次ラウンドで GPU がチェインの生成に利用する SP 群の決定
(2) 前ラウンドで GPU が生成したチェインをファイルに出力 (2 ラウンド目以降のみ)
(3) GPU がチェインの生成に利用する SP と重複しない

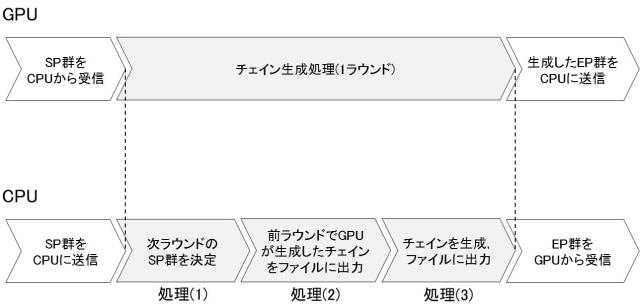

図 6 1 ラウンドあたりに CPU および GPU が行う処理
Fig. 6 Processing by CPU and GPU per round

SP をもとに CPU でチェインを生成

GPU でのチェイン生成処理 1 ラウンドあたりに CPU が行う処理のながれを図 6 に示す．処理 (1) では GPU が次ラウンドで利用する SP 群を決定する．SP 群は生成済みのチェインの SP と重複しないように決定する．処理 (2) では GPU が 1 回前のラウンドで生成したチェインを EP の先頭文字に応じて異なるファイルに分割して出力する．1 ラウンド目開始時点では生成済みのチェインは無いため，この処理は省略する．最終ラウンドでは GPU によるチェイン生成処理終了後，処理 (1)，処理 (3) を行わずに処理 (2) のみを行う．処理 (3) では GPU でのチェインを生成と並列して CPU で新たなチェインを生成する．処理 (2) でチェインを出力し終えたら処理 (3) を開始する．生成するチェインの SP には，既に生成されたすべてのチェインの SP と重複しない平文を用いる．CPU ではチェインを 1 本ずつ逐次的に生成し，1 本生成し終える毎に生成したチェインを EP に応じて異なるファイルに出力する．GPU でのチェイン生成処理が終了後，CPU で生成中のチェインが既定の長さを満たしたら CPU でのチェイン生成処理を終了する．

3.4 レインボーテーブルの分割

レインボーテーブルに保存されるチェインの総数は膨大である．レインボークラックではチェインの総数に比例して，TH と一致する EP の検索に要する時間は増加する．そこで EP の検索を効率化するために，EP の先頭の文字に基づいてレインボーテーブルを分割する方式を提案する．これにより，TH と先頭の文字が一致する EP が保存されたテーブルのみに検索対象を絞り込める．

レインボークラックでは TH と一致する EP が見つからなかった場合，TH をもとにチェインを生成し，新たに生成したハッシュ値と一致する EP を検索する．よって, 新たなハッシュ値が生成されるたびに，ハッシュ値の先頭の文字に対応したテーブルを検索対象として選択する必要がある．

表 1 実験環境
Table 1 Experiment environment

OS	CentOS 7.0
CPU	intel core i7-950 3.06GHz
ホストメモリ	16GB
GPU	NVIDIA Tesla K20c
デバイスメモリ	5GB
使用言語	c/c++
CUDA	CUDA 6.0
コンパイラ	g++/nvcc

なお，今回は EP の先頭 1 文字で分割したが，チェインの総数に応じ，n 文字に拡張することも同様に可能である．

4. 実験

4.1 実験内容

GPGPU による高速化の効果を評価するために逐次処理と提案手法によるそれぞれでレインボーテーブルの生成に要する時間を計測する．時間計測は最初の SP 決定処理の開始から，レインボーテーブル生成処理の終了までとする．生成したレインボーテーブルの性能を明らかにするために重複を除いた平文の網羅率を計測する．平文網羅率は式 (1) で表す．

$$\text{平文網羅率 (\%)} = \frac{\text{生成した平文の集合の要素数}}{\text{平文の総数}} \cdot 100 \quad (1)$$

算出した平文網羅率を用いて，生成したレインボーテーブルの性能を明らかにする．平文網羅率が高いほど解析成功率が高いレインボーテーブルであるといえる．

レインボーテーブルの分割の効果を明らかにするために，提案手法によって生成したレインボーテーブルと分割されていないレインボーテーブルを用いてレインボークラックを行う．ランダムに決定した 10,000 個のハッシュ値に対してレインボークラックを行い，合計所要時間を計測する．レインボークラックにおける TH と EP の一致判定を線形探索と 2 分探索の 2 通りで行い，所要時間を比較する．レインボークラックではテーブルをメモリに載せず，探索対象の EP が保存されているテーブルを随時選択しロードする．テーブルが分割されていない場合，テーブルを 1 度ロードした後は再ロードを行わない．

4.2 実験環境

評価実験における実験環境を表 1 に示す．表 1 に示した実験環境において，1 ラウンドあたりに GPU 上で生成可能なチェインの本数は 595,200 本である．

4.3 生成するレインボーテーブル

本研究では 4 文字の平文に対応したレインボーテーブルを生成する．平文に使用される文字は数字，英大文字，英小文字を合わせた 62 種である．平文のハッシュ化には Unix

の crypt(3) を用いる．crypt(3) が生成するハッシュ値に使用される文字はピリオド，スラッシュ，数字，英小文字，英大文字の 64 種である．

提案手法では生成したチェインを EP の先頭文字に応じて異なるファイルに保存する．crypt(3) が生成するハッシュ値の文字種は 64 種であるため，レインボーテーブルは 64 個のファイルに分割される．

4.3.1 ハッシュ関数

平文のハッシュ化に用いる crypt(3) は，任意長の平文に対して 11 文字のハッシュ値を生成するハッシュ関数である．crypt(3) の暗号化方式は DES(Data Encryption Standard)[6] である．DES は入力として与えられたビット列に並び替え処理を施すことによってビット列を暗号化する．crypt(3) は平文をもとに DES に用いる内部鍵を 16 個生成し，DES による並び替え処理を 16 回行う．並び替え処理によって生成されたビット列を文字列に変換し，ハッシュ値とする．

4.3.2 還元関数

チェインの生成に用いる還元関数を式 (2) に示す．

$$PT_i = \frac{RT}{{E_{PT}}^{i-1}} \bmod E_{PT} \ (0 \leq i < L_{PT}) \qquad (2)$$

式 (2) において，PT_i は平文の i 文字目，RT は還元する対象．E_{PT} は平文の文字種の数，L_{PT} は平文の文字数である．RT は還元対象のハッシュ値と平文の位置番号から生成される整数である．整数 RT の生成式を式 (3) に示す．

$$RT = HASH_{DEC} + P_{CHAIN} \cdot {E_{HASH}}^{L_{PT}} \qquad (3)$$

式 (3) において，$HASH_{DEC}$ は還元するハッシュ値を 10 進数の整数に置換した値，P_{CHAIN} はチェイン上における平文の位置番号，E_{HASH} はハッシュ値の文字種の数，L_{PT} は生成する平文の文字数である．式 (3) では，char 型の文字列であるハッシュ値を 10 進数の整数に変換する．本研究ではハッシュ値の文字種は 64 種類であるため，ハッシュ値を 64 進数の整数とみなして 10 進数の整数に変換した値を $HASH_{DEC}$ とする．

式 (2) による平文の i 文字目の生成では，RT に対して ${E_{PT}}^{i-1}$ による除算を行い，得られた剰余を PT_i とする．本研究では平文の文字種は 62 種類であるため，剰余は 0 から 61 までの整数である．よって，平文に使用される文字に 0 から 61 の整数を付与する．数字には 0 から 9，英大文字には 10 から 35，英小文字には 36 から 61 を ASCII 値の順番に沿って割り当てる．式 (2) の剰余と等しい整数を付与されている文字が平文の i 文字目として用いられる．

4.4 レインボーテーブル生成結果

1 ラウンドあたりに GPU が生成するチェインの本数を 595,200 本とし，圧縮前のチェインの長さが 10，30，50 のレインボーテーブルを提案手法によって生成した．

表 2 提案手法によって生成されたチェイン数
Table 2 Number of chains generated by the proposed method

ラウンド数	チェイン長 10	チェイン長 30	チェイン長 50
1	669,134	671,081	668,760
2	1,329,311	1,331,105	1,328,778
3	1,976,510	1,974,845	1,986,616
4	2,641,273	2,638,005	2,643,434
5	3,303,059	3,305,964	3,303,801
6	3,959,823	3,956,777	3,950,860
7	4,595,794	4,611,316	4,598,517
8	5,254,319	5,268,594	5,246,055
9	5,906,837	5,910,279	5,905,837
10	6,562,262	6,554,303	6,555,183

表 3 CPU で生成されたチェイン数
Table 3 Number of chains generated by CPU

ラウンド数	チェイン長 10	チェイン長 30	チェイン長 50
1	73,934	75,881	73,560
2	138,911	140,705	138,378
3	190,910	189,245	201,016
4	260,473	258,005	262,634
5	327,059	329,964	327,801
6	388,623	385,577	379,660
7	429,394	444,916	432,117
8	492,719	506,994	484,455
9	550,037	553,479	549,037
10	610,262	602,303	603,183

生成したレインボーテーブルに記録されているチェインの総数を実行ラウンド数ごとに表 2 に，CPU 上で生成されるチェイン数を表 3 に，テーブルの生成時間を表 4 に示す．CPU 上で生成されたチェインの本数は GPU がチェイン生成処理に要する時間に依存するため，同一条件下でレインボーテーブルを 10 回生成したさいの平均本数である．実行ラウンド数と生成したレインボーテーブルの平文網羅率の関係を表 5 に示す．GPU を用いた並列処理，CPU のみを用いた逐次処理によるチェイン長 50 のレインボーテーブル生成時間を表 6 に示す．

4.5 レインボークラック実行結果

生成したレインボーテーブルを用いて，ランダムに決定した 10,000 個のハッシュ値に対してレインボークラックを行った．レインボーテーブルはチェイン長が 50，チェイン生成処理の実行ラウンド数が 1 回，3 回，5 回のものを用いる．分割したテーブルと分割していないテーブルのそれぞれを用いて，線形探索と 2 分探索を行った場合の合計所要時間をそれぞれ表 7，表 8 に示す．ここで，解析成功率とは 10,000 個のハッシュ値に対して，パスワード解析が成功したハッシュ値の数の割合を表す．線形探索の場合，レインボーテーブル内のチェインは生成順であり，ソート

表 4　提案手法によるレインボーテーブル生成時間
Table 4　Generation time of rainbow tables by the proposed method

ラウンド数	チェイン長 10	チェイン長 30	チェイン長 50
1	11.20 s	40.34 s	1 m 10.64 s
2	20.61 s	1 m 20.37 s	2 m 21.28 s
3	31.15 s	2 m 00.30 s	3 m 21.87 s
4	42.12 s	2 m 41.20 s	4 m 42.51 s
5	50.95 s	3 m 21.37 s	5 m 53.56 s
6	1 m 01.30 s	4 m 01.24 s	7 m 04.32 s
7	1 m 11.44 s	4 m 42.01 s	8 m 15.15 s
8	1 m 23.59 s	5 m 22.40 s	9 m 25.83 s
9	1 m 33.09 s	6 m 03.31 s	10 m 36.46 s
10	1 m 42.42 s	6 m 43.96 s	11 m 47.04 s

表 5　生成したレインボーテーブルの平文網羅率 (%)
Table 5　Coverage of plaint text of generated rainbow tables

ラウンド数	チェイン長 10	チェイン長 30	チェイン長 50
1	36.99	62.31	77.61
2	53.02	81.42	91.12
3	66.73	89.91	95.98
4	74.59	94.64	96.90
5	79.23	96.16	98.13
6	84.94	97.71	98.57
7	87.69	98.24	99.09
8	90.37	98.53	99.23
9	92.62	98.93	99.42
10	93.11	99.10	99.60

表 6　逐次処理および並列処理による生成時間 (チェイン長：50)
Table 6　Generation time by sequential and parallel processing (Chain length：50)

ラウンド数	逐次処理	並列処理
1	11 m 21.63 s	1 m 10.64 s
2	21 m 40.46 s	2 m 21.28 s
3	32 m 12.28 s	3 m 21.87 s
4	42 m 30.72 s	4 m 42.51 s
5	53 m 55.35 s	5 m 53.56 s
6	63 m 52.11 s	7 m 04.32 s
7	74 m 01.42 s	8 m 15.15 s
8	83 m 48.49 s	9 m 25.83 s
9	95 m 16.52 s	10 m 36.46 s
10	105 m 59.83 s	11 m 47.04 s

表 7　レインボークラックの合計所要時間 (線形探索)
Table 7　Total time of Rainbow cracks (liner search)

ラウンド数	分割有り	分割無し	解析成功率 (%)
1	7 m 06.62 s	356 m 10.55 s	76.06
3	12 m 43.31 s	584 m 55.02 s	94.37
5	18 m 27.92 s	821 m 33.07 s	97.80

されていない．

表 8　レインボークラックの合計所要時間 (2 分探索)
Table 8　Total time of Rainbow cracks (binary search)

ラウンド数	分割有り	分割無し	解析成功率 (%)
1	3 m 16.46 s	3 m 25.77 s	76.06
3	4 m 32.11 s	4 m 46.63 s	94.37
5	5 m 40.07 s	6 m 00.39 s	97.80

5. 考察

提案手法によるレインボーテーブルの生成時間は実行ラウンド回数に正比例して増加した．これはラウンド数に関わらず各ラウンドで行われている処理が同一であるためである．

平文網羅率はチェインの長さと実行ラウンド回数に伴って増加した．これはレインボーテーブルの生成過程で生成される平文の総数がチェインの長さと実行ラウンド回数に依存するためである．1 本のチェインに含まれる平文の総数はチェインの長さに等しい．提案手法によって生成されるチェインの長さは均一であるため，生成するチェインの長さが n 倍になると，生成される平文の総数は n 倍になる．また，生成されるチェインの総数は実行ラウンド回数に正比例する．よって生成される平文の総数はチェインの長さと実行ラウンド回数に正比例する．生成される平文の総数が多いほど平文網羅率が上がりやすいが，平文網羅率の増加率はチェインの長さと実行ラウンド回数の増加に伴って低下した．これは生成される平文の総数が多いほど平文の重複が発生しやすくなるためである．平文網羅率が増加するほど網羅していない平文の総数は減少するため平文の重複が発生しやすくなり，網羅率の増加率が減少する．増加率の低下を防ぐには，平文の衝突がより発生しにくい還元関数をレインボーテーブルの生成に用いることによって衝突の発生回数を削減する必要がある．

GPU と CPU で並列してチェインを生成した結果，1 ラウンドあたりに CPU 上で生成されるチェインの本数は 60,000 から 70,000 本程度となった．高速化の余地として CPU でのチェイン生成処理を OpenMP などによって並列化することが挙げられる．

逐次処理と提案手法による並列処理のそれぞれでのレインボーテーブル生成時間を比較し，表 1 に示した実験環境においては並列処理の方が最大で約 9.7 倍速いことを確認した．これはレインボーテーブル生成処理の大部分を占めるチェイン生成処理を GPU を用いて並列処理したためである．GPU で大量のスレッドによる並列処理を行っているにも関わらず，高速化の効果は約 9 倍程度に収まっている．これは GPU での平文のハッシュ化において，各スレッドで char 型文字列とビット列の変換を行うためである．ハッシュ化では平文を char 型文字列からビット列に変換後，ハッシュ値として新たなビット列を生成し，生成し

たビット列を char 型文字列に変換する．文字列の変換では文字種に応じて処理が異なり，条件分岐が発生する．条件分岐が発生すると 32 スレッドの集合であるウォープの分岐方向が一致しなくなり，ウォープダイバージェントが発生し，処理速度の低下の原因となる [7].

生成したレインボーテーブルを用いて，線形探索と 2 分探索の 2 通りの方法でレインボークラックを行った．線形探索によって TH と一致する EP を探索する場合，提案手法で生成したテーブルを用いた場合の解析時間はテーブルを分割していない場合と比較して最大で約 50 倍速くなった．また，テーブル生成時のラウンド数が少ないほど効率化の効果が表れた．これはラウンド数が少ないほど平文網羅率が低くなり，解析の失敗率が高くなるためである．解析が成功すると TH と EP の一致判定は途中で終了するが，解析が失敗する場合はテーブル上のすべての EP と一致判定を行う．よって，解析成功率が低い時ほどすべての EP と一致判定を行う回数が増えるため効率化の効果が表れやすい．2 分探索によって TH と一致する EP を探索した結果，提案手法による効率化の効果は薄く，10,000 個のハッシュ値を解析したときの合計解析時間が最大で 5%程度速くなった．これは，2 分探索を行う対象をテーブル分割によって絞り込むことによって時間計算量が削減されたが，探索対象のテーブルを TH を更新する毎に読み込む必要があるため，効率化の効果が薄まったと考えられる．

6. 関連研究

Graves の研究 [8] では CUDA を用いて NTLM ハッシュに対応したレインボーテーブルを生成している．Graves の提案手法ではチェイン生成処理は GPU のみで行われている．提案手法によってチェインの総数 100，チェインの長さ 100,000 のレインボーテーブルを生成した結果，生成時間は 18 分 50 秒としている．

Gómez らの研究 [9] では MPI(Message Passing Interface)，CUDA のそれぞれで総当たり攻撃とレインボーテーブルの生成を行っている．実験の結果，総当たり攻撃では CUDA が有利としているが，レインボーテーブルの生成では MPI が有利としている．

7. おわりに

本稿では GPGPU を用いたレインボーテーブルの生成手法を提案し，レインボーテーブルの生成を高速化した．また，TH と EP の一致判定を効率化するためにレインボーテーブルの分割手法を提案した．生成したレインボーテーブルの性能を平文網羅率を算出することによって明らかにした．

評価実験において，提案手法を用いて数字，英小文字，英大文字によって構成される 4 文字のパスワードに対応したレインボーテーブルを生成した．生成したレインボーテーブルはチェインの長さ 50，チェイン数 4,598,517 の場合において平文網羅率は 99.09 %となった．提案手法によるレインボーテーブルの生成時間は CPU のみによる逐次処理と比較して最大で約 9.7 倍速い．また，テーブル分割によるレインボークラックの効率化の効果を明らかにするために，生成したレインボーテーブルを用いて線形探索と 2 分探索のそれぞれでクラックを行った．レインボークラックを行った結果，線形探索では分割されていないテーブルを用いた場合と比較して解析時間が最大で約 50 倍速くなった．2 分探索の場合は提案手法による効率化の効果は薄かった．

今後の課題として，さらに長い文字長の平文に対応したレインボーテーブルを生成するために，複数台の GPU でチェインを並列生成することが挙げられる．また，GPU でのチェイン生成処理を高速にするために，ハッシュ化において発生しているウォープダイバージェントの削減が挙げられる．CPU 上で SP をビット列に変換する処理を行い，GPU 上でのチェイン生成処理では平文をすべてビット列として扱うことによってウォープダイバージェントの削減が可能となり，高速化率の向上が期待できる．

謝辞 本研究の一部は JSPS 科研費 15K00112 および 2016 年度南山大学パッヘ研究奨励金 I-A-2 の助成による．

参考文献

[1] 桑門秀典，森井昌克：総当たり攻撃に対して安全な認証関数の構成法，情報処理学会論文誌，vol.50，No.9，pp.1930-1941，(2009).

[2] RainbowCrack Project：RainbowCrack，入手先〈http://project-rainbowcrack.com/〉，(2016/08/01).

[3] 安藤公希，桑原寛明，上原哲太郎，國枝義敏：MPI 並列処理によるレインボーテーブル生成の高速化，コンピュータセキュリティシンポジウム 2015 論文集，vol.3，pp.1335-1342，(2015).

[4] 安田幹，佐々木悠：暗号学的ハッシュ関数—安全神話の崩壊と新たなる挑戦，電子情報通信学会 基礎・境界ソサイエティ Fundamentals Review，Vol.4，No.1，pp.57-67，(2010).

[5] NVIDIA：CUDA Parallel Computing Platform，入手先〈http://www.nvidia.com/object/cuda_home_new.html〉，(2016/08/01).

[6] NIST Computer Security Resource Center：DATA ENCRYPTION STANDARD (DES)，入手先〈http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf〉，(2016/08/01).

[7] 加藤誠也，須田礼二，玉田嘉紀：GPU におけるダイバージェンス削減による高速化手法，研究報告ハイパフォーマンスコンピューティング（HPC），vol.2012-HPC-134(5)，pp.1-11，(2012).

[8] Russell Edward Graves：High performance password cracking by implementing rainbow tables on nVidia graphics cards (IseCrack)，Master's thesis，Iowa State University，(2008).

[9] Julio Gómez，Francisco G Montoya，R Benedicto, et al：Cryptanalysis of Hash Functions Using Advanced Multiprocessing，Distributed Computing and Artificial intelligence，pp.221-228，Springer，(2010).