ブロックチェーンによる乱数生成の透明性確保

江原 友登[1]　多田 充[2]

概要：近年，乱数はゲームやオンライン抽選など様々な用途で使用され，私たちの生活にも多く関わっている．しかし，そうした乱数による結果が本当に乱数による結果なのかを私たちユーザ側が知る手段がなく，結果に納得できないこともある．示された結果がサーバ側の意思が働いていない，正当な手順で正しく作成された乱数によるものだとユーザが確認できることが重要である．そこで，本論文ではブロックチェーンの特徴である透明性のあるデータ管理機能を利用した，透明性のある乱数生成方法について提案し，実装，検証を行うことを目的とする．

キーワード：ブロックチェーン，乱数，イーサリアム

How to secure transparency for random number generation using blockchain

Yuto Ehara[1]　Mitsuru Tada[2]

Abstract: Nowadays, random numbers become to have various use applications such as games and on-line lots, and become so related to our lives. However, we are often unconvinced with some results by random numbers, since we have no means knowing whether they are indeed the results derived by random numbers. It is important that users can verify that the results shown have come via random numbers correctly generated with legitimate processes and without servers' intentions. Then, this paper presents how to generate random numbers with transparency using transparent management functions being one of the properties of blockchains.

Keywords: Blockchain，Random number, Ethereum

1. はじめに

近年，インターネット，さらにはスマートフォンの普及により，オンラインゲームのガチャやネットでの抽選など乱数が使用されているコンテンツをほとんどの人が利用したことがある．しかし，そうしたコンテンツの結果が本当に正当な手順によって作成された乱数によって得られた結果なのかをユーザが確認することは基本的にはできない．そのため，結果に納得できないユーザもいる．納得してもらうためには，示された結果がサーバ側の意思が働いていない，正当な手順で正しく作成された乱数によるものだとユーザが確認できることが重要である．

公平に生成された乱数であっても，結果だけを見ていると，一般的な人間には不自然に見えてしまうことは多い．佐古らは，ハッシュ関数をランダムオラクルとみなした暗号学的に安全な乱数発生メカニズムをベースに，ブロックチェーンの透明性という特徴を活かし，バックギャモンというゲームにおいてゲーム終了後に使用された乱数を検証する方式を提案した．しかし，この方式ではサーバーが信用できる第三者であることが前提とされており，ユーザと

[1] 千葉大学大学院 理学研究科
〒 263-8522 千葉市稲毛区弥生町 1-33
Graduate School of Science, Chiba University.
Yayoicho 1-33, Inage, Chiba 263-8522, Japan.
e-mail: acka1834@chiba-u.jp
[2] 千葉大学 統合情報センター
Institute of Media and Information Technologies, Chiba University.
e-mail: m.tada@faculty.chiba-u.jp

サーバが結託すれば，不正をすることが可能である．

そこで，本稿ではブロックチェーンを利用し，トラストレスな状況でも正当な方法で乱数生成を行い，その乱数列を検証する方式を提案する．また，そのシステムをブロックチェーンを用いた分散アプリケーションプラットフォームの Ethereum に実装することで，乱数生成のスマートコントラクトを作成し，乱数の一様性評価やコスト評価を行うことを目的とする．

本稿の内容は以下のとおりである．第 2 節では，ブロックチェーン関連の技術について説明する．第 3 節では，ブロックチェーンを用いた乱数生成方法について，これまでに発表されている既存研究について説明する．第 4 節では，提案方式及び実装方法について述べる．第 5 節では，考察として提案方式の利点・欠点，実装評価について述べる．最後に，第 7 節でまとめを述べる．

2. ブロックチェーン関連技術

本章では，ブロックチェーン技術の概略，それに付随する用語の定義，および，その関連技術について述べる。

2.1 ブロックチェーンの仕組み

2008 年に Satoshi Nakamoto と名乗る人物が「Bitcoin: A Peer-to-Peer Electronic Cash System」[7] と題された論文を発表した．文献 [7] によって生み出されたのが「ビットコイン」という仮想通貨（暗号通貨）であり，このビットコインを支える技術として，文献 [7] に書かれているものがブロックチェーンという技術である．ブロックチェーンとは世界中に点在するコンピューターにデータを分散することで，中央集権を置かずにデータを破壊・改ざん・偽装が困難なネットワークを作る技術であり，従来の集中管理型のシステムに比べ，

(1)『改ざんが極めて困難』である
(2)『実質ゼロ・ダウンタイム』なシステムが実現できる
(3)『安価』に構築可能である

という特性を持つといわれている [1], [8].

各ブロックには

- いつそのブロックが作られたのかを証明するタイムスタンプ
- 前のブロックハッシュ値
- ナンス（Nonce）と呼ばれる値（後に説明）
- 一定時間でのトランザクション（取引）情報

が含まれ，次のブロックへと時系列でチェーンのように伸びていく．

2.1.1 マイニング

ブロックチェーンにおいて新しいブロックを作成することをマイニング（採掘）と呼ぶ．特定の条件を満たしたマイナー（採掘者）が，トランザクションの検証を行いで正当性が確認できたトランザクション等を上述のようなデータ

図 1 ブロックチェーンのデータ構造

のブロックを作成し，ブロックチェーンネットワークにブロードキャストする．マイニングに成功したマイナーは，インセンティブとして，そのブロックに含まれるトランザクションの取引手数料や新しく発行される仮想通貨を得られるのが一般的である．基本的にブロックチェーンネットワークに参加しているすべてのノードがマイナーになる権利を持っている．マイナーとして，マイニングを成功させるためには，「コンセンサスアルゴリズム」という特定の条件を満たす必要がある．これについては以下に述べる．

2.1.2 コンセンサスアルゴリズム

コンセンサスアルゴリズムとは，P2P ネットワークのように情報到達にタイムラグがあるネットワークにおいて，参加者が単一の結果について相互承認するためのアルゴリズム一般を指す．コンセンサスアルゴリズムの一種が Proof of Work である．コンセンサスアルゴリズムとしては他に，Proof of Stake や Proof of Importance などがあるが，現在多く使われているのがこの Proof of Work である．直訳すると「仕事の証明」となり，単純だが手間がかかる，ただし本当にそれを行ったことの検証は簡単な，特定の作業あえて行わせることにより，悪意のないことを確認する（不正を行う動機を低減させる）」という仕組みである [8].

具体的な手順としては次のようである．マイナーは，自分の手もとに届いている正当性が確認できたトランザクションデータの集合や 1 つ前のブロックハッシュ値，タイムスタンプ等にナンスと呼ばれる任意の値を加えてハッシュ値を計算していき，得られたハッシュ値が設定された値（difficulty target）より小さくなるようなナンスを探す．そのようなナンスを最初発見したマイナーが新しいブロックを作成することができる．ハッシュ関数は一方向性があるため，総当りで計算を続けるしかないため，ナンスを操作・予測することは困難であり，さらに言えば得られるハッシュ値（ブロックハッシュ値）も同様である．

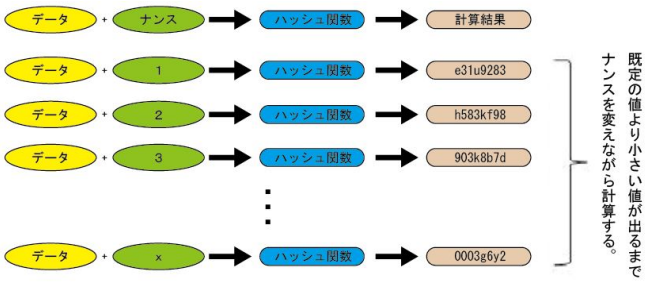

図 2 ナンスの求め方

2.1.3 チェーンの枝分かれ

ブロックチェーンにおいてはほぼ同時に二つのノードが Proof of Work を成功させるなどして一時的に複数のブロックが枝分かれすることがある．その場合，それ以降により早く一定数のブロックがつながったほうを正当なものとして判定する．ビットコインでは 6 ブロック程度が生成されれば正当なブロックとしてみなすことが慣習となっている．

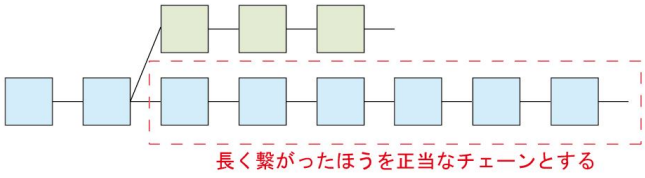

図 3　チェーンの枝分かれ

2.1.4 ブロックチェーンへの攻撃

ブロックチェーンは，過去の情報を保持した状態でつながれていくため，コンセンサスアルゴリズムが Proof of Work である場合，攻撃者がトランザクションの改ざんや偽装を行うためには

(1) 改ざんするデータを含んだブロック以降のブロックを全て作り直す

(2) 正当な枝分かれよりも早くブロックを成立させ続ける

などの方法が考えられるが，（1）の場合 以下同様過去のブロックのナンス値をすべて計算しなおさないといけなく，(2) の場合でもブロックチェーンネットワークに参加する全てのコンピュータの計算能力の 50%より大きな割合を確保する必要があり，どちらにしても大量の計算能力が必要であり現実的でない．逆に言えば 50 %以上の計算能力を保持すれば不正できるということであり，この問題は 51 %攻撃と言われている．しかし，50%より大きな計算能力を確保できたとしても，その場合正当にマイニングを行ったほうが経済合理性が高いため，不正が行われにくいとされる [8].

2.2 Ethereum[2], [5], [11]

Ethereum（イーサリアム [2], [5], [11]）は 2015 年にリリースされた次世代型スマートコントラクト分散アプリケーション基盤であり，スイスを拠点とした Ethereum Foundation により開発が進められているオープンソースプロジェクトである．「ether」という仮想通貨が実装されており，ビットコインと違い，単に仮想通貨のやり取りだけでなくユーザが独自に定義したスマートコントラクトを実行することができる．2017 年現在，Proof of Work が採用されているブロックチェーンを用い，マイナーがマイニングを行うと，取引の実行として通貨の送金やスマートコントラクトに記述したプログラムを実行し，記録することで正当性を担保する．

Ethereum は EOA（Externally Owned Account）と CA(Contract Account) という二つのアカウントの種類を持つ．EOA はユーザが使用するアカウントで，通貨の送金やコントラクトを実行をすることができる．CA はコントラクトをブロックチェーンにデプロイした際に作成されるアカウントで，EOA や CA からメッセージを受け取ると記述されたコードを実行する．

マイナーによって CA は実行される．CA を実行するようにリクエストした EOA はマイナーに対し，「Gas」と呼ばれる実行手数料をインセンティブとして支払う．支払う実行手数料は，次の要素で決定する．

Gas Fee：コード実行量によって決まる，コード実行に必要な gas の量．単位は gas.

Gas Price：1Gas あたりの価格．単位は wei/gas (1 ether = 10^{18}wei).

支払う実行手数料は，これら二つをかけ合わせた値である．

コントラクトのコードの記述は Solidity[4] と呼ばれるスクリプト言語が一般的に使用される．本研究では Solidity を使用し，実装を行った．

3. 既存研究

本章では，ブロックチェーンを用いた乱数生成方法について，これまでに発表されている [9]，および，本論文とは独立に考えられている [10] について述べる．

3.1 オンラインゲームへの応用

SCIS2017 において佐古らが「ブロックチェーンを用いたオンラインゲーム用公平性を検証可能な乱数発生」[9] という論文を発表している．文献 [9] では，バックギャモンというオンラインゲームにおける乱数について，ビットコインブロックチェーン上に記録をすることでユーザに操作したものではなく公平な乱数によるものであることを納得してもらうことを行った．しかし，この論文の方式では，データ通信をサーバを介して行っているため，サーバがあるプレーヤーと結託すると不正が行われてしまう可能性がある．そのため，トラストレスな状況でも正当な乱数生成として本方式を提案する．

3.2 Ethereum を用いた乱数生成コントラクト

2017 年に書籍 [10] が発行された。この文献 [10] の第 6 章において，Ethereum を用いた乱数生成コントラクトについて述べられており，公平な乱数生成を実現している．ただし，本論文の提案方式は，これとは独立に考察されたものである [*1]。

[*1] 文献 [10] は 2017 年 8 月 3 日に初版第 1 刷が発行されているが，本論文 (CSS2017) の発表申込〆切は 2017 年 8 月 2 日である。

4. 提案方式

本章では，提案方式，および，その実装方法について述べる。なお，実装による評価については第 5 章で述べる。

4.1 登場エンティティの定義

本方式に登場するエンティティは以下の通りである．

- $U_1, \ldots, U_N$：ユーザ
- S：サービス提供者
- Gen：乱数生成者
- E：ブロックチェーン保持者

各ユーザ U_i は，ID に相当するアカウントアドレス a_i を持つが，提案方式ではユーザ同士の通信が発生しないので，表記を簡潔にするため，ユーザを U，そのアカウントアドレスを a と表記する．乱数生成者 Gen が実行するアルゴリズムは，全てのエンティティに知られているものとする．ブロックチェーン保持者 E は，信頼できるエンティティとし，少なくとも各 U および S が参照できるブロックチェーン BC を持ち [*2]，以下ように，クエリ $\mathtt{Reg}, \mathtt{AskNonce}$ に対して処理/応答をするものとする．

$\mathtt{Reg}(\mathrm{info})$：情報 info を受け付け，その info を BC に登録する．

$\mathtt{AskNonce}()$：BC において，次に作られるブロック内のナンス n を返す [*3]．

4.2 乱数生成の要件

オンライン抽選において望ましい乱数生成方式に求められる要件は以下の通りである．

(A) 出力の値に関して，抽選を行うユーザとサービス提供者が関与できること

(B) 出力の値に関して，全ユーザ，サービス提供者が予測操作できないこと

(C) 出力された値が正当なものであったことを全ユーザ，サービス提供者共に確認できること

これらに加えて，可能であればユーザの操作が楽であることが望ましい．

4.3 提案方式

第 4.2 節で示した要件を満たす方式を以下のように構成する．なお，関数 Hash は適切なハッシュ関数とする。

[step 1] 抽選を行いたいユーザ U は，乱数 r を生成する [*4]．

[step 2] U は抽選を行うことを乱数生成者 Gen にリクエストし，このとき引数として乱数 r を渡す．(このリクエストを req（$=(a, r, \gamma)$）とする。γ は補助的な情報であり，空でもよい．)（図 4 の①）

[step 3] Gen は，リクエスト req を受けたら，一意的な整理券番号 sn を発行し，その後，sn を U に返し，サービス提供者 S に整理券番号 sn が与えられているリクエストが来たことを通知する．（図 4 の②）また，Gen は，$\mathtt{Reg}(\mathsf{sn}, \mathsf{req})$ を実行し，$(\mathsf{sn}, a, r, \gamma)$ を BC に登録する．（図 4 の③）

[step 4] S は，通知（sn）が来たら，sn に対応する乱数 p を作成し，Gen に送信する．（図 4 の④）

[step 5] Gen は p を受け取ったら，$\mathtt{AskNonce}()$ を実行し（図 4 の⑤），次のブロックが作成されたタイミングでそのブロックのナンス n を E から入手し（図 4 の⑥），$\mathtt{Reg}(p, n)$ を実行し，p と n を BC に登録する．（図 4 の⑦）

[step 6] Gen は $\alpha = \mathtt{Hash}(r||p||\mathsf{sn}||a||n)$ を計算し（図 4 の⑧），この値 α を U と S に送信する．（図 4 の⑨）

[step 7] U と S は，送られてきた結果（α）が，r, p, sn, a と n に基づいて得られた結果なのかどうかを確認する．

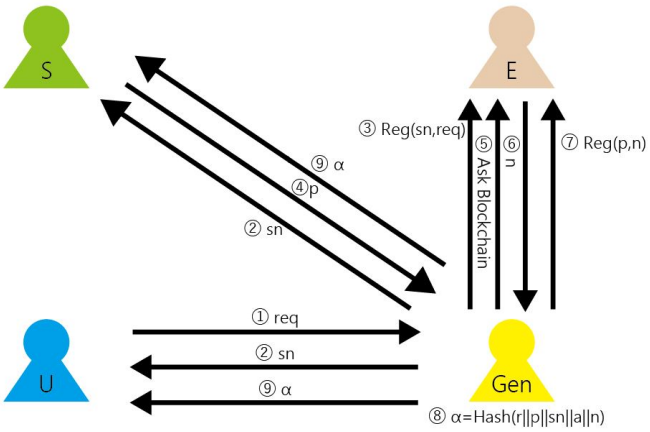

図 4　提案方式

4.4 要件充足について

上記の提案方式は，第 4.2 節における乱数生成の要件を満たしている。実際，以下の通りである。

要件 (A)：出力される値は U，S が選んだ r，p に依存する。

要件 (B)：出力の結果 α は，ナンス n の値に依存しており，ナンスの値は一般的に予測できない。

要件 (C)：[step 7] による。

また，ユーザは乱数 r の生成と検証に関与するだけであり，

[*2] Ethereum におけるブロックチェーンのように，一般公開されている (パブリック) ブロックチェーンであれば十分である．

[*3] AskNonce については，次にブロックが作られるのを待たなければならにので，クエリを受け付けてから応答を返すまでに，若干の時間がかかる．第 5 章における実装実験では，Ethereum のテスト環境であることもあり，返答までに約 5〜10 秒かかっている．実際の Ethereum の場合は，返答に平均 15 秒かかるように，ナンスの difficulty target が調節されている [5], [11].

[*4] 乱数 r の定義域 R は，適切なビット数 k に対して $\{0,1\}^k$ としてよい。本論文では，R(および k) の値は本質的ではないので，以降，各エンティティが生成する乱数の定義域については，その記述を省略する。

その際，何かしらの選択をしたり，意思表示する必要がないので，面倒な操作はほとんどないと言える．

4.5 実装について

今回は第 4.1 節におけるブロックチェーン保持者 E として Ethereum を採用し実装を行った．また，乱数生成者 *Gen* も Ethereum に担わせることとした．

基本方式に沿った CA を付録 A.1 のように実装した．なお，今回の実装では通知送信等の通信部分に関しては本質的な部分ではないため実装していない．

5. 考察

本章では，本論文の提案方式の利点・欠点，既存研究との比較，実装評価について述べる．

5.1 提案方式の利点・欠点

提案方式の利点は，実装の際，実際に運用されているブロックチェーンを利用することにより，トラストレスな状況でも検証可能な（透明性のある）乱数生成を可能としている点である．

サービス提供者単体，ユーザ単体，またはユーザやサービス提供者が結託しても，出力される値を予測・操作することは困難である．これは，まだ採掘されていないブロックのナンス（実装ではブロックハッシュ値）を採用しているためである．逆に言えば，ブロックのナンスを予測することが可能になれば，本方式では公平性を保てないことになるが，ハッシュ関数を適切に選ぶことにより，ナンスを見つけるためには総当たりでハッシュ計算を行わなければらなくなり，現時点では困難であると思われる．

また，ブロックが採掘されてから新しいブロックが採掘されるまでのリクエストは全て同じナンスを利用するため，ユーザ複数とサービス提供者側で協力することでユーザ複数が同じ値を出力させることが可能であるが，出力させる値には整理券番号やユーザアドレスにも依存しているため，同じ値を作成することは非常に困難であると言える．もし，同じ値を作成することができたとしても出力される値を予測することはできないので予測困難性という点では問題はない．他にも，*Gen* が実行するアルゴリズムが参加エンティティに知られている，つまり，乱数生成のコードをユーザも確認できることが可能であることや，出力される値に自分の入力する値が依存していることから，ユーザは納得感を得やすいと思われる．実装においては Ethereum を用いているが，Ethereum で実行されるプログラムはブロックチェーン上に記録されており，全てのエンティティが確認できる．

欠点としては，1 回抽選をするごとに操作が必要なため，1 人のユーザが連続で抽選する場合などのユーザビリティの問題が挙げられる．また，提案方式における乱数生成を，ゲームにおけるクジ引き例えると，このクジ引きにおいては，クジを引きその結果を確認した後，引かれたクジを元に戻すことになる．したがって，何度クジを引いても同じ値が出たり，逆に特定の（望む）結果が得られなかったりする．生成された乱数の大小関係等を利用したり，生成された乱数の値に対して異なる法を適用することにより，予め当たりくじの種類や本数が決められている（一度引かれたクジを戻さない）クジ引きを実現することができる [*5]．

5.2 既存研究との比較

文献 [9] との差は，Ethereum などの実際に運用されているブロックチェーンを利用することにより，トラストレスな状況でも正当な乱数生成が可能であるという点である．まだ採掘されていないブロックのナンス（実装ではブロックハッシュ値）を利用することで，サービス提供者も出力する値を予測・操作することが困難になっている．そのため，文献 [9] においてはサービス提供者側があるユーザと結託して，違うユーザを陥れることが可能であるが，本提案方式ではそれを防ぐことができる．文献 [10] との差に関しては，ユーザ，サービス提供者共に出力される値に関与できる点である．ユーザおよびサービス提供者が関与してもしなくても，ハッシュ関数を適切に選ぶことにより，出力される結果に対する乱数性の問題はなくなるが，ユーザが出力値に関与することにより，納得感を得られることが期待できる．

5.3 実装評価

提案方式により生成される乱数が一様性を確保できているかどうかを検証するために，以下の実装実験を行った．

第 4.3 節で述べた提案方式におけるパラメータ等については以下の通りである．$r, p,$ `sn` は長さ 256 のビット列，アカウントアドレス a は長さ 160 のビット列であり，γ は空列である．また，ハッシュ関数 `Hash` は SHA256 である．

コインの裏表と 100 分立の抽選を想定して，付録 A.1 に掲載するソースコードにおける `MaxNumber` の値を 2 および 100 とし，それぞれの場合について試行回数を 10000 回として乱数生性を行った．

ユーザとサービス提供者が関与する乱数に関しては，JavaScript の構文で「Math」オブジェクトの「random()」メソッドを使用し，値域 1〜100000 の間の整数をランダムで入力とした．結果はそれぞれ図 5 のようになっている．どちらも生成された整数値は均等に発生していることが確認できる．

χ^2 検定を行い，p 値を求めたところ，`MaxNumber=2` の場合は 0.779278，`MaxNumber=100` の場合は 0.673676 となり，どちらも有意水準の値 0.05 よりも大きいため，出力値

[*5] ライブの席決めの抽選などは，このようにして実現できる．

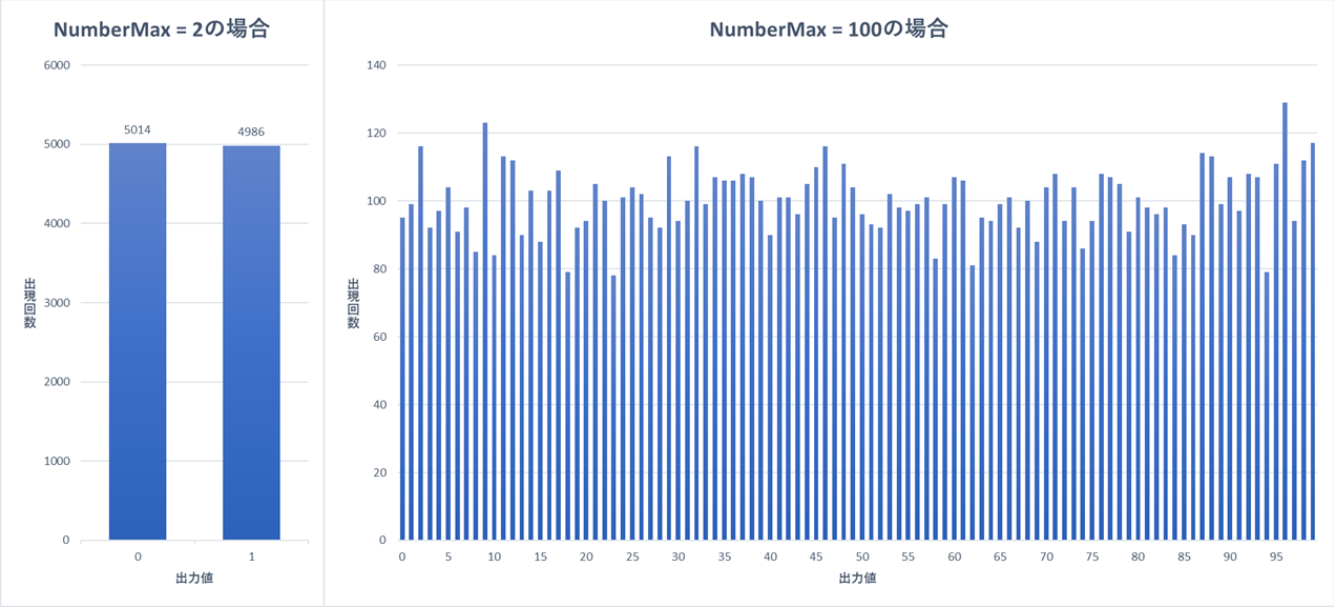

図 5 実装結果

は十分にばらついていることが証明できた．

提案したシステムを用いて，抽選を行った場合にどの程度のコストがかかるのかを評価する．第 2.2 節に述べた通り，CA を実行するごとにコードの実行量によって Gas が必要になる．なお，GasPrice は 2017 年 8 月現在の平均レート [6] を用い，1Gas あたり約 0.000000023ether であった．1ether あたりのレート [3] は 2017 年 8 月現在約 323 ドルである．これらのレートに従い，第 4.5 節の通り実装をすると，デプロイするときに約 5 ドル，1 回抽選を行うごとに約 1 ドル（関数 draw, give 共に約 0.5 ドル）必要であった．

6. まとめ

本論文では，ブロックチェーンの特徴である透明性のあるデータ管理機能を利用した，透明性のある乱数生成方法について提案し，Ethereum において実装，検証を行った．Ethereum を用いることで，トラストレスな状況でも正当な方法で乱数生成を行えるスマートコントラクトを可能とし，乱数としてばらつきがあることや 1 回の抽選で約 1 ドルのコストがかかることを示した．今後の課題としては，ユーザビリティの向上，difficulty target と不正難易度の関係を明らかにしていくことなどが挙げられる．

参考文献

[1] 赤羽，愛敬：ブロックチェーン 仕組みと理論，リックテレコム，2016.

[2] V. Buterin: “*White paper: A next-generation smart contract and decentralized application platform*”, wiki on GitHub (Self-published), 2015.
`https://github.com/ethereum/wiki/wiki/White-Paper`
(2017 年 8 月 25 日閲覧)

[3] CoinMarketCap: “Ethereum (ETH) price, charts, market cap, and other metrics”.
`https://coinmarketcap.com/currencies/ethereum/`
(2017 年 8 月 25 日閲覧)

[4] Ethereum: “Solidity”.
`https://solidity.readthedocs.io/en/develop/`
(2017 年 8 月 25 日閲覧)

[5] Ethereum Foundation: “Ethereum project”, 2017.
`https://www.ethereum.org/`
(2017 年 8 月 25 日閲覧)

[6] Etherscan: “Ethereum Average GasPrice Chart”.
`https://etherscan.io/chart/gasprice`
(2017 年 8 月 25 日閲覧)

[7] Satoshi Nakamoto: “*Bitcoin: A peer-to-peer electronic cash system*”, 2008.
`https://bitcoin.org/bitcoin.pdf`
(2017 年 8 月 25 日閲覧)

[8] 株式会社野村総合研究所：「我が国経済社会の情報化・サービス化に係る基盤整備（ブロックチェーン技術を利用したサービスに関する国内外動向調査）報告書」
`http://www.meti.go.jp/press/2016/04/20160428003/20160428003-2.pdf`
(2017 年 8 月 25 日閲覧)

[9] 佐古，井口：「ブロックチェーンを用いたオンラインゲーム用公平性を検証可能な乱数発生」，2017 年暗号と情報セキュリティシンポジウム (SCIS2017), 1F2-4, 2017.

[10] 渡辺，松本，西村，清水：はじめてのブロックチェーンアプリケーション Etereum によるスマートコントラクト開発入門，翔泳社，2017.

[11] G. Wood: “*Ethereum: A secure decentralised generalised transaction ledger, EIP-150 revision*”, Self published, 2014.
`http://gavwood.com/paper.pdf`
(2017 年 8 月 25 日閲覧)

付　　録

A.1 実装コードについて

ここでは，第 4.5 節で述べた実装コードを示す．

Owner はサービス提供者のアドレス，MaxNumber は欲しい乱数値の値域設定パラメータ，NumberedTicket は整理券番号パラメータであり，Information は整理券番号に紐づけられる乱数を管理するデータ構造である．関数 draw は，[step 1][step 2][step 3] にあたるものであり，ユーザが抽選をするときに使う．関数 give は，[step 4][step 5] にあたるものであり，draw が実行された後，サービス提供側が使う．関数 refer は，[step 5][step 6] にあたるものであり，出力結果を参照する．また，提案方式ではナンスを用いているが，ナンスを参照するプロパティが現在なかったため，ナンスと同様に予測・操作が困難であるブロックハッシュ値を用いた．

```
pragma solidity ^0.4.8;
contract RNG{
    address Owner;
    uint MaxNumber;
    uint NumberedTicket;

    struct Information{
        uint BlockNumber;
        address QueryAddress;
        bytes32 Seed;
        bytes32 OSeed;
    }

    mapping (uint => Information) Query;

    modifier onlyOwner() {
       if (msg.sender != Owner) throw; _;
     }

    function RNG(uint _max){
        Owner = msg.sender;
        MaxNumber = _max;
        NumberedTicket = 0;
    }

    function draw(bytes32 _seed) returns(uint,
         bytes32){
        Query[NumberedTicket].QueryAddress =
            msg.sender;
        Query[NumberedTicket].Seed = _seed;
        uint _box = NumberedTicket;
        NumberedTicket = NumberedTicket + 1;
        return (_box, _seed);
    }

    function give(uint _index, bytes32 _oseed)
         onlyOwner returns(int status, uint
        index, bytes32 seed2){
       if(_index >= NumberedTicket){
            return(-2, _index, 0);
        }else{
            if(Query[_index].OSeed != 0){
                return(-1, _index, 0);
            }else{
                Query[_index].BlockNumber =
                    block.number;
                Query[_index].OSeed = _oseed;
                return(0, _index, _oseed);
            }
        }
    }

    function refer(uint _Index) constant
        returns(int status, address ad, uint
        index, bytes32 seed1, bytes32 seed2,
        bytes32 blockhash, bytes32 grn, uint
        dn){
        uint _i = _Index;
        if(_i >= NumberedTicket){
            return(-3, Query[_i].QueryAddress,
                 _i, 0, 0, 0, 0, 0);
        }else{
            if(Query[_i].OSeed == 0){
                return(-2, Query[_i].
                    QueryAddress, _i, 0, 0, 0,
                     0, 0);
            }else{
                uint _next = Query[_i].
                    BlockNumber + 1;
                if (_next > block.number){
                    return(-1, Query[_i].
                        QueryAddress, _i, 0,
                        0, 0, 0, 0);
                }else{
                    bytes32 _blockhash = block
                        .blockhash(_next);
                    bytes32 _grn = sha256(
                        _blockhash, Query[_i].
                        QueryAddress, Query[_i
                        ].Seed, Query[_i].
                        OSeed,  _i);
                    uint _dn = uint(_grn) %
                        MaxNumber;
                    return(0, Query[_i].
                        QueryAddress, _i,
                        Query[_i].Seed, Query[
                        _i].OSeed, _blockhash,
                         _grn, _dn);
                }
```

```
            }
        }
    }
}
```