ブロックチェーンを用いた検証可能な抽選システムの提案

廣澤 龍典[1,a)]　上原 哲太郎[2,b)]

概要：近年，インターネットやスマートフォンの普及により，オンラインサービスにおける抽選の機会が多くなっている．しかし，抽選に用いられる乱数の生成手法や抽選結果がその乱数の生成結果を正しく反映しているかは基本的に確認できない．そのため，ユーザは抽選の結果に納得できない場合がある．そこで我々は，ブロックチェーンを用いた検証可能な抽選システムを提案する．ブロックチェーンによって乱数の生成手法や乱数と賞品の対応付けは記録・公開されるため，ユーザはブロックチェーンに記録されたデータから抽選の正当性を確認することが可能となる．

キーワード：ブロックチェーン，Ethereum，乱数，抽選

Verifiable loot box system using blockchain

Tatsunori Hirosawa[1,a)]　Tetsutaro Uehara[2,b)]

Abstract: With the spread of the Internet and smartphones, there have been many opportunities to draw lots in online services. We often want to verify the fairness of such lots as the operator might cheat. However, it is basically impossible to confirm whether the random number generator is working normally and the results reflect the random numbers accordingly under the published probabilities. Therefore, we propose a verifiable loot box system using blockchain. The players can also confirm the validity of the lots from the corresponding data recorded on the blockchain.

Keywords: Blockchain,Ethereum,Random number,Loot box

1. はじめに

近年，インターネットやスマートフォンの普及により，オンラインサービスにおける抽選の機会が多くなっている．例えば，SNS を利用した懸賞では，ユーザはシステムと擬似的なジャンケンやトランプによって勝敗を決め，勝てば賞品を受け取ることができる．また，ゲームにおいては，そのゲーム内において使用できるアイテムを抽選で配布している．しかし，オンラインサービスにおける抽選は，現実の抽選より不正を行いやすい．運営者の不正事案の例として，2018 年 1 月，希少なキャラクターが当たる確率を実際より高く見せかけたとして，ゲーム会社「アワ・パーム・カンパニー・リミテッド」が再発防止の為の措置命令を受けた [1]．特定のキャラクターが 3%の確率で当たると告知していたが, 実際は 0.33%であった．

このような抽選では，希少度の高いアイテムを獲得するために個人が高額のお金を使うことがある[*1]．よって，サービス運営者が不正を行い，生成された乱数やその乱数と賞品の対応付けを偽っていないかどうか，その正当性を担保する必要がある．しかし，抽選に用いられる乱数の生成手法やアイテムが公表された確率通りに配布されているかは，基本的に確認できない．そのため，ユーザは抽選の結果に納得できない場合がある．そこで本稿では，ブロックチェーンを用いた検証可能な抽選システムを提案する．

[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] 立命館大学 情報理工学部
College of Information Science and Engineering, Ritsumeikan University
[a)] hirosawa@cysec.cs.ritsumei.ac.jp
[b)] t-uehara@fc.ritsumei.ac.jp

[*1] スクウェア・エニックスが提供する「星のドラゴンクエスト」のガチャでは 90 万円の課金額を支払ったプレイヤーが存在する [2]

2. 研究背景

2.1 ブロックチェーン

2.1.1 概要

ビットコインは，2008 年に Satoshi Nakamoto が発表した論文 “Bitcoin: A Peer-to-Peer Electronic Cash System”[3] にて提唱された暗号資産（仮想通貨）である．ビットコインではブロックチェーンという仕組みを利用している．ブロックチェーンは，一種の分散データベースと考えることができる．従来の分散データベースと比べたブロックチェーンの特徴は

- 改ざんが極めて困難であること
- 中央集権的なサーバを必要とせずにデータの整合性を確保できること
- 記録されているデータを誰でも閲覧できること

の 3 点である．

ブロックチェーンの構造を**図 1** に示す．ブロックチェーンは，名称どおりブロックをチェーンのように時系列に従って繋いでいる．ブロックとはデータの塊であり，図に示したように

- トランザクションと呼ばれるデータの最小単位
- ブロック作成日時を示すタイムスタンプ
- 前のブロックのハッシュ値
- ナンス（後述）

などの情報を含んでいる．

2.1.2 マイニングとコンセンサスアルゴリズム

ブロックチェーンに新たなブロックを繋げる作業をマイニング（採掘）と呼び，マイニングを行う者をマイナーと呼ぶ．マイニングを行う際にはコンセンサスアルゴリズムを遵守する必要がある．

コンセンサスアルゴリズムとは，複数の参加者の間で全体の合意を取る手続きのことである [4]．ビットコインでは，Proof of Work と呼ばれるコンセンサスアルゴリズムを用いることで，ブロックチェーンに繋ぐ新たなブロックの合意を取っている．具体的な手順は以下の通りである．

(1) マイナーは P2P ネットワークから取得したトランザクションのうち，正当性が検証できたトランザクションを選択する
(2) 選択したトランザクションに加えて，タイムスタンプ・前のブロックのハッシュ値・ナンスと呼ばれる任意の値を合わせてハッシュ値を計算する
(3) 得られたハッシュ値が規定の値より小さいか（ハッシュ値の先頭に 0 がいくつあるか）を確認する
(4) 規定を満たしていれば完成したブロックを P2P ネットワークに送信し，他のマイナーに確認してもらった上でブロックチェーンに繋げる
(5) 規定を満たしていなければナンスを変更し，もう一度 (2) のハッシュ計算を行う
(6) (3) 〜 (5) を繰り返す

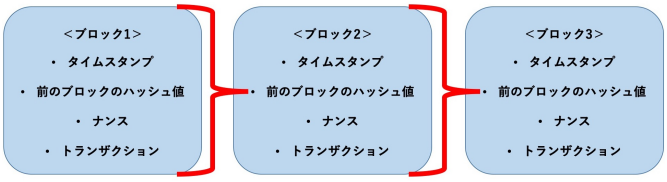

図 1 ブロックチェーンの構造
Fig. 1 Blockchain structure

ハッシュ関数は一方向性を持つため，規定を満たすハッシュ値からナンスを逆算することはできない．つまり，マイナーは規定を満たすハッシュ値を見つけるためにはナンスを総当たりしなければならない．よって，ナンスおよび完成するブロックのハッシュ値の予測操作は困難である．

Proof of work は得られたハッシュ値が規定の値より小さいかのみを成否の条件としており，唯一の正解は存在しない．そのため，複数のマイナーが条件を満たすブロックを完成させ，ブロックチェーンに繋げることがある．この場合，一時的に枝分かれ（フォーク）が発生する．枝分かれが発生した場合は，それ以降により早く一定数のブロックが繋がったブロックチェーンを正当なものとする．

2.2 Ethereum と Solidity

ビットコインは暗号資産をやり取りすることに主眼を置いているため，カスタマイズ可能な領域が少ない．そのため，暗号資産以外の領域での利用は困難である．この問題を Ethereum（イーサリアム）と呼ばれる，分散アプリケーション向けのグローバルなオープンソース・プラットフォーム [5]（ブロックチェーン基盤）が解決した．Ethereum は 2013 年にその構想がホワイトペーパー [6] で示され，現在は Ethereum Project によって開発・運営が行われている．Ethereum においては暗号資産「ether」が実装されており，ビットコインと同じように，ブロックチェーンを利用することで取引の正当性を保証している．

ビットコインと異なり Ethereum は，専用のプログラミング言語を用いることで，スマートコントラクトを記録・実行できる．スマートコントラクトとは，契約にまつわるさまざまなリスクや課題を情報技術で解決しようとする考え方である [4]．Ethereum におけるスマートコントラクトは，ブロックチェーン上に記録され実行されるプログラムのことを指す．スマートコントラクトは，暗号資産のやりとり同様にマイナーがマイニングを行うことで，ブロックチェーンに記録される．ブロックチェーンに記録する際には記録者を表すアドレスも一緒に記録される．このため，ブロックチェーンを閲覧するツールを用いることで，誰でもスマートコントラクトの記録者や内容の正当性，動作内容の検証を行うことが可能である．また，スマートコントラクトを実行する過程でブロックチェーンへ情報を記録し

た場合は，記録内容と記録者がブロックチェーンに記録され，誰でもその内容を確認できる．

Solidity は Ethereum においてスマートコントラクトを記述するためのプログラミング言語の一つである．Solidity の特徴として

- C++，Python，JavaScript の影響を受けて設計されていること
- 静的型付言語であること
- 継承やライブラリ，ユーザ定義型をサポートしていること

がある [4]．また，Solidity ではデータの保存方法として storage と memory の二種類が存在する．storage とはデータをブロックチェーンに記録するものであり，半永久的にデータを記録する．storage としてブロックチェーンに記録したデータは誰でも閲覧することができる．memory はスマートコントラクトの実行時に，一時的にデータを保持するものである．memory として扱ったデータはブロックチェーンに記録されない．スマートコントラクトを作成する際には，ブロックチェーンに記録すべきデータは stoage で扱い，実行のために用いるだけで良いデータは memory で扱う．以上のように，Solidity ではスマートコントラクト特有の設計や実装もあるものの，int や string などの変数をはじめとし，順次処理・分岐処理・反復処理といったプログラムの基礎的な動作が可能である．

Solidity によって記述されたスマートコントラクトは，Ethereum Virtual Machine と呼ばれるスマートコントラクトの実行環境において，バイトコードに変換した上で実行される．具体的には，ユーザはスマートコントラクトの実行を求めるトランザクションを生成し，ネットワークにブロードキャストする．トランザクションを取得したマイナーは，その内容に従いスマートコントラクトを実行する．この時，ユーザは「Gas（単位は gas)」と呼ばれる実行手数料をマイナーに対し ether で支払わなければならない．支払う Gas は，1gas あたりの価格とスマートコントラクトのコード実行量に比例した gas の量で計算される．ただし，ブロックチェーンに情報を書き込まない（状態遷移を伴わない）プログラムは，マイニングおよび Gas を支払うことなく実行することができる．

2.3 抽選システムに対する規制の現状

商品及び役務の取引に関連する不当な景品類及び表示による顧客の誘引を防止するために施工されている，不当景品類及び不当表示防止法（景品表示法）の第五条は，不当な表示の禁止について定めている [7]．また，消費者庁が公開した資料 [8] には，オンラインゲームにおけるガチャに対する景品表示法の規制について詳細が記されている．

業界団体はオンラインゲームにおけるガイドラインを定めている [9], [10]．これらのガイドラインでは，ガチャで獲得できる賞品とそれらの提供確率の表示，ガチャで配布する賞品の価値の上限，提供確率の不適切な変更の禁止などをルールとして定めている．

また，Apple は App Store で提供するアプリケーションに対するガイドラインを定めている [11]．ガイドラインの 3.1 項には “「ルートボックス」などの方法でバーチャルアイテムをランダムに購入できる App では、各種アイテムの入手確率を明記して、ユーザーが購入前に確認できるようにしてください。”とアイテムの入手確率の明記を義務付けている．ガイドラインを満たさないアプリケーションを App Store で配信することはできない．

これらの法律やガイドラインは抽選の過程を公にすることを義務付けていないという問題点がある．例えば 1 %の確率で提供される賞品があったとして，そのように確率を表示した場合でも，100 回に 1 回当たればこの条件は満たすことができる．そのため，極端な例ではあるが賞品が当たるタイミングを恣意的に 100 回目の抽選にしたとしても，表示した確率を守ることが可能となる．有償の抽選では 1 回目の抽選で当たる場合と 100 回目の抽選で当たる場合では必要なコストに差が発生する．

2.4 関連研究

佐古らは SCIS2017 において「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」[12] という論文を発表している．この論文では，バックギャモンと呼ばれるオンラインゲームにおいて，生成される乱数をビットコインのブロックチェーン上に記録している．そして，ブロックチェーンに記録された乱数を検証することで，ユーザに対して生成された乱数が公平なものであると納得してもらおうとしている．しかし，この手法はサーバを介した通信を行なっているため，ユーザとサーバが結託することで不正が成立してしまうという課題点が存在する．

江原らは CSS2017 において，前述の問題を克服を目指し「ブロックチェーンによる乱数生成の透明性確保」[13] という論文を発表している．この論文では，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している．また，抽選の透明性を確保し，抽選の結果に納得してもらえるよう，乱数を生成する際のシード値にユーザ・サービス運営者がそれぞれ作成した乱数を含めている．

渡辺らは「はじめてのブロックチェーン・アプリケーション Ethereum によるスマートコントラクト開発入門」[14] という書籍の第 6 章において，スマートコントラクトを用いた乱数生成について考察を行い，公平な乱数生成手法を考案している．第 6 章の最終節では，スマートコントラクト外の情報（WolframAlpha）を用いることで，より簡易に乱数生成を実現する方法も紹介している．

江原ら・渡辺らは共に未採掘ブロックのハッシュ値を乱

数生成のシード値に用いることで，ユーザ・サービス運営者共に生成される乱数を予測操作困難にした．未採掘ブロックのハッシュ値を使う理由は，ユーザ・サービス運営者共に抽選結果の操作を防ぐためである．たとえマイナーであっても，総当たり計算によってナンスを求めるため，新たに作られるブロックのハッシュ値を予測操作できない．しかし，完成（公開）されている最新のブロックのハッシュ値をシード値として乱数を生成すると，あらかじめ乱数生成を試しておき，自身が望む乱数が出たタイミングで抽選を行うことで，結果を操作できてしまう．その反面，その時点で作られていないブロックのハッシュ値を用いると，そのブロックが完成するまでブロックのハッシュ値は誰にも分からない．つまり，あらかじめ乱数生成を試しておき，結果を操作することはできなくなる．未採掘ブロックのハッシュ値を用いるためには，抽選システムはユーザが抽選を要請してきた時点で完成している最新のブロックの番号をユーザに渡すと共にシステム内に控えておく．そして，ユーザは次のブロックが新たに採掘された時点で，もう一度抽選システムに対して抽選の要請を行う．抽選システムは控えてあった番号と照会を行い，確かにその時点で抽選をしようとしたことを確認する．そして，抽選システムは一度目の要請時点で未採掘だったブロックのハッシュ値を基に，乱数を生成することが可能となる．

3. 提案手法

3.1 コンセプト

関連研究では，乱数を生成する上でユーザ・運営者共に未採掘のブロックのハッシュをシード値に含めることで，生成される乱数を予測操作不可能としていた．そのほか，同じタイミングであってもユーザ毎に結果が変わるように「ユーザ毎に異なるものとしてユーザアドレス」，同じユーザであっても抽選 1 回毎に結果が変わるように「抽選を行なった数」をシード値の一部としていた．また，生成された乱数が正当なものであったことをユーザ・サービス運営者共に確認できるよう，乱数を生成する際のシード値をユーザ・サービス運営者の一方のみが知ることがないようにしていた．これらは公平さを保つために踏襲する．

本研究における変更点は

(1) ユーザは乱数値の生成の一部に関与できる
(2) サービス運営者はシード値に関与しない
(3) ルートボックス（ガチャ）を模して乱数と賞品の対応付けをスマートコントラクト内で行う

の 3 点である．

(1) について，ユーザはサービス運営者と比べて不利な立場にあるため，任意の文字列をシード値に含めるようにする．シード値には未採掘ブロックのハッシュ値も含まれている以上，ユーザが文字列によって特定の抽選結果を操作することはできない．しかし，シード値に関与できるようにすることでユーザの納得感が向上すると考える．これは，ガラガラ抽選機などにおいて，本来の方向と逆に回してガラガラの中の玉を混ぜる行動と似ている．ユーザは抽選機から出てくる玉の色を操作することはできないが，この行動によってユーザは抽選結果に納得感を得ることができる．

(2) について (1) とは反対に，サービス運営者は乱数生成のプログラムを作成しサービスを運営する性質上，有利な立場にある．そのため，ユーザが抽選結果を操作できない限りは過度にシード値に関与する必要はない．そして，未採掘ブロックのハッシュ値を使用することで，ユーザは抽選結果を操作することはできないようになっている．よって，サービス運営者はシード値に関与しないようにする．

(3) について乱数と賞品の対応付けをスマートコントラクトで実装する．これにより，サービス運営者が乱数は正しく生成したが，対応付けにおける抽選結果を偽るという不正を防ぐことができる．また，1 回の抽選で，複数回分の抽選結果を得られるようにする．これはゲームなどにおける「10 連ガチャ」に相当するものであり，10 回分の抽選結果を得るために 10 回の抽選を行う場合と比べて実行時間の短縮が図れる．

3.2 提案手法の詳細

乱数生成および賞品との対応付けのステップを**図 2** に示す．図中の番号に基づき詳細を説明する．

① サービス運営者は乱数の最大値を決め，スマートコントラクトをデプロイする．
② サービス運営者は各賞品の確率を決め，設定する．
③ 抽選を行いたいユーザは自身の好きな文字列をスマートコントラクトに送信し，整理券をリクエストする．
④ スマートコントラクトは，ユーザが送信した文字列・リクエスト送信時点のブロック番号を記録し，整理券

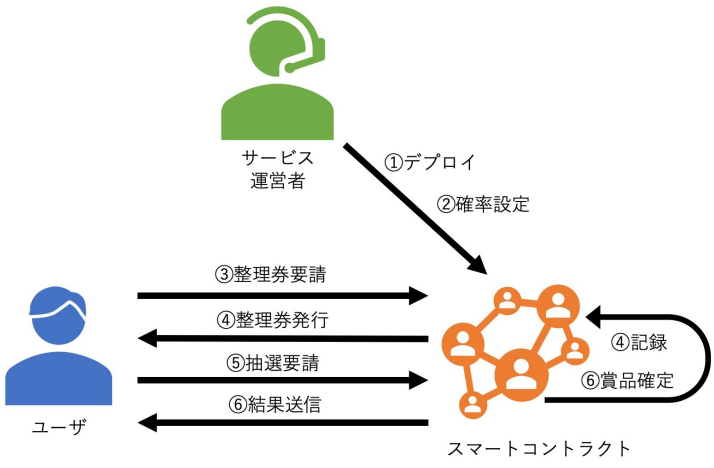

図 2 提案手法
Fig. 2 Proposed method

をユーザに返す．

⑤ ユーザは一定時間経過後に整理番号と抽選を行う回数をスマートコントラクトに送信する．

⑥ スマートコントラクトは整理券に基づいて「未採掘だったブロックのハッシュ値・ユーザのアドレス・整理券番号・ユーザが指定した文字列・1 回の抽選内において何回目であるかを示すカウンタ」をシード値として乱数を生成する．生成した乱数がどの賞品に該当するか確認し，賞品名を返す．

3.3 実装方法

ブロックチェーンおよびスマートコントラクトのプラットフォームは Ethereum を選択した．スマートコントラクト開発用 IDE として Remix[*2]を用いた．テスト環境用ブロックチェーン（プライベートネットワーク）の構築には Geth[*3]を用いた．Geth のバージョンは 1.8.27 である．また，スマートコントラクトを記述する際のプログラミング言語は Solidity を使用し，そのバージョンは 0.4.26 である．プログラムの全容は付録 A.1 に掲載したとおりである．

4. 実験

4.1 実験（1）

4.1.1 実験内容

実験（1）では抽選結果を複数得たい場合のユーザビリティの検証と，乱数と賞品の対応付けが可能であるかを確認した．生成される乱数の最大値を 1,000 に設定し，乱数と賞品の対応付けとして，3 種類（Normal，Rare，Ultimate）の賞品があると仮定し実験を行なった．それぞれの賞品を獲得できる確率は 50%，30%，20%とした．そして，1 回の抽選要請で 10 回分の抽選結果を獲得できるかを確認した．ユーザが送信する文字列は “Ritsumeikan” である．また，スマートコントラクトをデプロイおよび実行する際にかかったコスト（Gas）についても確認を行なった．

4.1.2 実験結果

乱数生成および賞品の獲得結果を**図 3** に示す．図

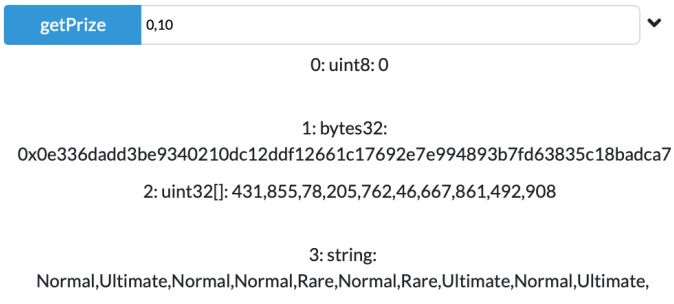

図 3 実験結果

Fig. 3 Experimental result

*2 https://github.com/ethereum/remix-ide

*3 https://geth.ethereum.org

表 1 デプロイ・関数実行時のコスト

Table 1 Exection cost

	Gas	Ethereum 換算	日本円換算
デプロイ	1,418,927 gas	0.01477026156 eth	360.24 円
setProbability	41,049 gas	0.00042729785 eth	10.42 円
request	89,848 gas	0.00093526901 eth	22.81 円
getPrize	0 gas	0 eth	0 円
stringConnect	0 gas	0 eth	0 円

中に示されている値は上から，ステータスコード（失敗時に 1 以上が返される）・ブロックハッシュ・生成された乱数・乱数に対応する賞品である．今回の抽選では乱数が「431,855,78,205,762,46,667,861,492,908」が生成され，それに対応する賞品として「Normal,Ultimate,Normal,Normal,Rare,Normal,Rare,Ultimate,Normal,Ultimate」が獲得できた．

また，実験 1 においてスマートコントラクトを実行するにあたって発生したコストを**表 1** に示す[*4]．実験時の為替レートは 0.000000010409458393 ETH/gas，および 24,389.84 JPY/ETH であった[*5]．

4.2 実験（2）

4.2.1 実験内容

乱数の一様性が確保できているか確認するため，10 種類の文字列を用いてそれぞれ 1,000 回の乱数生成を行った．生成される乱数の最大値は 65536 に設定した．使用した文字列は “Cirrusnic”，“Cirrocumulus”，“Cirrostratus”，“Altocumulus”，“Altostratus”，“Nimbostratus”，“Stratocumulus”，“Stratus”，“Cumulus”，“Cumulonimbus” の 10 個である．生成される乱数の範囲を 16 等分し，それぞれの期待度数が 625 であるとし，生成した計 10,000 個の乱数について，χ^2 検定を行い p 値を求めた．

4.2.2 実験結果

χ^2 乗検定の結果，p 値は 0.7635 となった．実験結果を散布図にまとめたもの**図 4** に示す．生成された乱数を 256 で割った商と剰余をそれぞれ X 軸・Y 軸に取ったものである．前述の 10 種類の文字列ごとに生成された乱数を 2 種類ずつ 1 つの系列（1 系列あたり 2000 個の点）にまとめてプロットした．散布図において塊は見受けられない．

5. 考察

5.1 実行時間とコスト

4.1.2 で示したとおり，1 回の抽選で 10 個の乱数とこれに対応する賞品を 10 個獲得できているため，オンラインゲームなどで実装されている「10 連ガチャ」と同じ挙動が再現できた．既存研究では，1 回の抽選にて得られる乱数

*4 日本円換算に関しては小数第 3 位を四捨五入した

*5 https://etherscan.io/chart/gasprice および https://coinmarketcap.com/currencies/ethereum/より

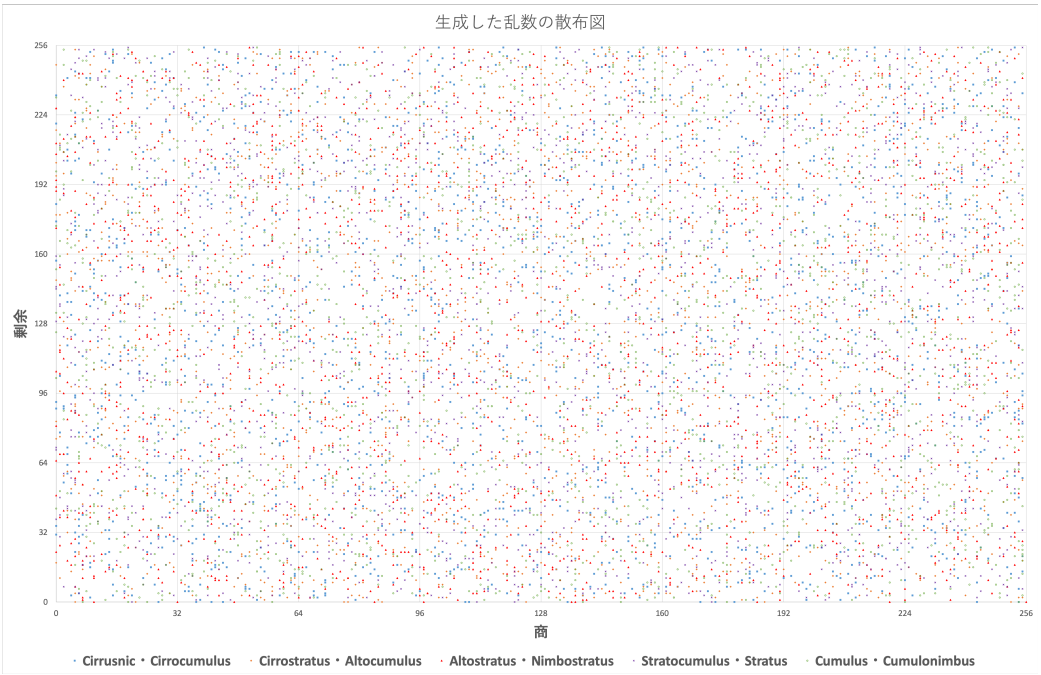

図 4 生成した乱数の散布図

Fig. 4 Scatter diagram of generated random numbers

は 1 個限りだったため，本提案手法で実行時間の短縮が図れたといえる．

次に，デプロイおよび関数の実行にかかったコストについては，為替レートによって変動するものの，執筆時点では現実的なコストであると考える．また，getPrize 関数および stringConnect 関数は状態遷移を伴わずに乱数生成と賞品の対応付けを行なっているため，実行する際にコストは不要である．そのため，1 回の抽選要請によって得ようとする賞品の個数が増えてもコストが上がることはない．つまり，獲得する賞品が多いほど，抽選を要請する際のコストが実質的に下がるといえる．これはユーザの射幸心を煽るデメリットも生じるものの，運営者にとっては 1 回あたりの抽選で得られる賞品の個数が多ければ多いほど付加価値（割引など）を付けやすくなり，ユーザにとってもメリットは存在する．

5.2 乱数の一様性

一様性について p 値が 0.7635 であり有意水準である 0.05 より大きいため，生成した乱数は十分にばらついているといえる．また，ユーザが指定した好きな文字列を入れた場合でも未採掘ブロックの性質およびハッシュ関数の性質上，シード値を操作することができないため，公平性と納得感を両立したと考えている．

6. おわりに

本稿ではブロックチェーンを用いた検証可能な抽選システムの提案を行なった．Ethereum をプラットフォームとし，スマートコントラクトで乱数生成および乱数と賞品との対応付けを行うことで正当な抽選システムを実現した．その結果，実行時間を短縮させたシステムを実装することができ，抽選システムを使用ためのコストも現実的なものであると示した．また，生成される乱数の一様性が確保できていることも示した．

今後の課題として，本抽選システムは一度生成した乱数を除外することなく，何度でも生成することができるようになっている．これはイベントなどにおける座席の抽選等では不適切である．そのため，本システムが対応できる抽選の種類を増やすことが課題だと考える．また，暗号資産の価値や採掘難易度と，抽選で得られる賞品の価値のバランスによっては，不正なブロックを作成した上で抽選結果の操作をし賞品を獲得した方が得する場合もありうる．そのため，暗号資産の価値と賞品の価値のバランスを考察する必要がある．

参考文献

[1] 消費者庁：アワ・パーム・カンパニー・リミテッドに対する景品表示法に基づく 措置命令について，入手先<https://www.caa.go.jp/policies/policy/representation/fair_labeling/pdf/fair_labeling_180126_0001.pdf >（参照 2019-7-28）．

[2] ねとらぼ：スクエニ「星のドラゴンクエスト」ガチャ不当表示で集団訴訟に 1 人で 90 万円以上課金したユーザーも，入手先<https://nlab.itmedia.co.jp/nl/articles/1801/26/news113.html >（参照 2019-7-28）．

[3] Satoshi Nakamoto：Bitcoin: A Peer-to-Peer Electronic Cash System，入手先<https://bitcoin.org/bitcoin.pdf >（参照 2019-7-28）．

表 A·1 関数とステップの関連
Table A·1 Associating functions and steps

関数	ステップ
constructor	①
setProbability	②
request	③・④
getPrize	⑤・⑥
stringConnect	⑥

[4] 加嵜 長門, 篠原 航, 金 志京, 河西 紀明, 田中 克典, 佐々木 亮彰, 平野 浩司, 前川 彰, DMM.com ブロックチェーン研究室：試して学ぶ スマートコントラクト開発, マイナビ出版 (2019).

[5] Ethereum Project:Home — Ethereum, 入手先<https://www.ethereum.org > (参照 2019-7-28).

[6] Ethereum Project:A Next-Generation Smart Contract and Decentralized Application Platform, 入手先<https://github.com/ethereum/wiki/wiki/White-Paper > (参照 2019-7-28).

[7] 電子政府の総合窓口 e-Gov：不当景品類及び不当表示防止法 (景品表示法), 入手先<https://elaws.e-gov.go.jp/search/elawsSearch/elaws_search/lsg0500/detail?lawId=337AC0000000134 > (参照 2019-7-28).

[8] 消費者庁：オンラインゲームの「コンプガチャ」と景品表示法の景品規制について, 入手先<https://www.caa.go.jp/policies/policy/representation/fair_labeling/guideline/pdf/120518premiums_1.pdf > (参照 2019-8-6).

[9] 一般社団法人 コンピュータエンターテインメント協会：ネットワークゲームにおけるランダム型アイテム提供方式運営ガイドライン, 入手先<https://www.cesa.or.jp/uploads/2016/release20160427.pdf > (参照 2019-7-28).

[10] 日本オンラインゲーム協会 ガイドラインワーキンググループ：ランダム型アイテム提供方式を利用したアイテム販売における表示および運営ガイドライン, 入手先<https://japanonlinegame.org/wp-content/uploads/2017/06/JOGA20160401.pdf > (参照 2019-7-28).

[11] Apple：App Store Review ガイドライン - Apple Developer, 入手先<https://developer.apple.com/jp/app-store/review/guidelines/ > (参照 2019-7-28).

[12] 佐古 和恵, 井口 圭一：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

[13] 江原 友登, 多田 充：ブロックチェーンによる乱数生成の透明性確保, コンピュータセキュリティシンポジウム 2017 (CSS2017), 2E4-4, 2017

[14] 渡辺 篤, 松本 雄太, 西村 祥一, 清水 俊也：はじめてのブロックチェーン・アプリケーション Ethereum によるスマートコントラクト開発入門, 翔泳社 (2017).

付　　録

A.1 ソースコード

実装したスマートコントラクトのソースコードを以下に示す. 関数と 3.2 にて述べたステップの関連を**表 A·1** に示す.

```
pragma solidity ^0.4.26;

contract RandomNumber{
    address owner;
    uint32 maxNumber;

    uint16 normal;
    uint16 rare;
    uint16 ultimate;

    struct lootBox{
        uint blockNumber;
        string phrase;
    }

    struct lootBoxes{
        uint16 numberedTicket;
        mapping(uint16 => lootBox)lootBoxes;
    }
    mapping(address => lootBoxes)requests;

    event TicketAndPhrase(uint16 numberedTicket,
        string phrase);
    event ReturnProbability(uint16 normal,uint
        rare,uint16 ultimate);


    modifier isOwner(){
        require(msg.sender == owner);
        _;
    }

    constructor (uint32 _max)public{
        owner = msg.sender;
        maxNumber = _max;
    }

    function setProbability(uint16 _normal,uint16
        _rare,uint16 _ultimate) isOwner public{
        normal = _normal;
        rare = _rare;
        ultimate = _ultimate;
        emit ReturnProbability(normal,rare,
            ultimate);
    }

    function request(string _phrase) public
        returns(uint16,string){
        requests[msg.sender].lootBoxes[ requests[
            msg.sender].numberedTicket ].
            blockNumber = block.number;
        requests[msg.sender].lootBoxes[ requests[
            msg.sender].numberedTicket ].phrase =
            _phrase;
        requests[msg.sender].numberedTicket++;

        emit TicketAndPhrase(requests[msg.sender].
            numberedTicket-1,_phrase);
        return (requests[msg.sender].
            numberedTicket-1,requests[msg.sender].
            lootBoxes[ requests[msg.sender].
            numberedTicket-1 ].phrase);
    }
```

```
51
52      function getPrize(uint16 _numberedTicket,uint8
            _numberOfTimes) public view returns(uint8
        ,bytes32,uint32[],string){
53          uint32[] memory randomNumber = new uint32
                [](_numberOfTimes);
54
55          if(_numberedTicket >= requests[msg.sender
                ].numberedTicket){
56              return(2,0,randomNumber,"Unknown␣
                    Numbered␣Ticket");
57          }else{
58              uint targetBlockNumber = requests[msg.
                    sender].lootBoxes[_numberedTicket
                    ].blockNumber;
59              targetBlockNumber++;
60              if(targetBlockNumber >= block.number){
61                  return(1,0,randomNumber,"Not␣Mined
                        ");
62              }else{
63                  string memory prize;
64                  string memory usersPhrase =
                        requests[msg.sender].lootBoxes
                        [_numberedTicket].phrase;
65
66                  for(uint8 i=0; i<_numberOfTimes; i
                        ++){
67                      bytes32 seedPhrase = keccak256
                            (abi.encodePacked(
                            blockhash(
                            targetBlockNumber),msg.
                            sender,_numberedTicket,
                            usersPhrase,i));
68                      randomNumber[i] = uint32(
                            seedPhrase) % maxNumber +
                            1;
69
70                      if(randomNumber[i] <= normal){
71                          prize = stringConnect(
                                prize,"Normal,");
72                      }else if(randomNumber[i] >
                            normal && randomNumber[i]
                            <= rare){
73                          prize = stringConnect(
                                prize,"Rare,");
74                      }else if(randomNumber[i] >
                            rare && randomNumber[i] <=
                             ultimate){
75                          prize = stringConnect(
                                prize,"Ultimate,");
76                      }
77
78                  }
79                  return(0,blockhash(
                        targetBlockNumber),
                        randomNumber,prize);
80              }
81          }
82      }
83
84      function stringConnect(string _prize,string
            _rarity) public pure returns(string){
85          bytes memory bytePrize = bytes(_prize);
86          bytes memory byteRarity = bytes(_rarity);
87          bytes memory connect = new bytes(bytePrize
                .length + byteRarity.length);
88          uint64 pointer=0;
89
90          for(uint64 i=0; i<bytePrize.length; i++){
91              connect[pointer++] = bytePrize[i];
92          }
93          for(uint64 j=0; j<byteRarity.length; j++){
94              connect[pointer++] = byteRarity[j];
95          }
96          return (string(connect));
97      }
98
99  }
```