mPoW: 意味のある計算によるProof of Work

浅沼 岳樹[1,a)]　五十部 孝典[1,b)]

概要：ブロックチェーンのコンセンサスアルゴリズムである Proof of Work(PoW) では，意味のない大量のハッシュ演算計算をベースにしており，電力や計算資源を無駄遣いしている．2020 年において，ビットコインの PoW に費やす電力はベルギーの年間消費電力（82TWh）に及び，持続可能な開発目標（Sustainable Development Goals; SDGs）に反する重大な問題である．この問題を解決するために本稿では，遺伝的アルゴリズム (Genetic Algorithm; GA) を PoW に組み込むことで PoW を意味のある演算にする Meaningfull PoW(mPoW) を提案する．具体的には，GA の最適化計算により周期的に生成される中間値をビットコインに用いられている Hashcash の入力として利用することにより，PoW に求められる特性を保ったまま意味のある演算（GA の最適化問題）が可能である．さらに，Device binding 技術と組み合わせることで，大量のメモリの利用を求めることなく ASIC 耐性を持たせることも可能であり，mPoW は消費電力と計算資源両方の無駄を削減できることを示す.

キーワード：ビットコイン, Proof of Work, 消費電力, Hashcash, 遺伝的アルゴリズム

mPoW: How to make Proof of Work Meaningful

Takaki Asanuma[1,a)]　Takanori Isobe[1,b)]

Abstract: Proof of Work (PoW), the consensus algorithm for blockchain, is based on a large number of meaningless hash calculation, wasting electric power and computational resources. In 2020, Bitcoin's PoW consumed as much electricity as the annual power consumption of Belgium (82TWh). This is a serious problem against the Sustainable Development Goals (SDGs). To solve this problem, this paper proposes Meaningful PoW (mPoW), which makes PoW a meaningful calculation by applying Genetic Algorithm (GA) to PoW. Specifically, by using the intermediate values that are periodically generated through GA calculations as input to the Hashcash used in Bitcoin, it is possible to make this scheme a meaningful calculation (GA optimization problem) while maintaining the properties required for PoW. Furthermore, by applying device-binding technology, mPoW can be ASIC resistant without requiring the large amounts of memory. Thus, we show that mPoW can reduce the waste of both power consumption and computational resources.

Keywords: Bitcoin, Proof of Work, Electricity consumption, Hashcash, Genetic Algorithm

1 兵庫県立大学大学院
a) takaki.asanuma@gmail.com
b) takanori.isobe@ai.u-hyogo.ac.jp

© 2021 Information Processing Society of Japan

1. はじめに

1.1 背景

サトシ・ナカモトが考案したビットコインの登場はブロックチェーンへの関心を集め，現在では研究や議論が国際的に盛んに行われる分野の 1 つである [1]．暗号資産においては，いかにして二重支払いを防ぐかが重要な課題である．単純な解決策としては，信頼のおける第三者の中央機関（銀行等）を介して取引すれば良い．しかしながら，この中央集権の解決策では，中央サーバーへの攻撃によりシステム全体が停止するリスクや，中央機関自体の不正利用等のリスクがある．そこで，ビットコインを代表とする非中央集権での暗号資産は，ブロックチェーンと呼ばれる仕組みを採用している．ブロックチェーンの基本的な考えは，正確な取引データの維持により二重支払いを防止しようとする方式である．具体的には，取引データを誰にでも見られる形で保存し，そのデータの改ざんが不可能なのであれば，二重支払いを阻止することが可能とする．

Proof of Work(PoW) は，ブロックチェーンの合意形成で用いられている基盤技術である [2]．PoW では，取引データの正当性の検証をネットワークの参加者による多数決に委ねることで，非中央集権型のシステムを実現しており，ブロックチェーンに保存されたデータの耐改ざん性を担保している．ビットコインなどのパブリックブロックチェーンにおいて，PoW により耐改ざん性を実現している．

1.2 既存の PoW の問題点

問題点 1：電力の無駄遣い

PoW が電力を無駄遣いしていることが社会問題として指摘されている [3–6]．ビットコインで適用されている Hashcash という PoW では，ネットワークの参加者にハッシュ値の原像復元問題を計算させているため，問題の内容としては意味のない計算を強制していることになる [7]．2020 年において，ビットコインの PoW に費やす電力はベルギーの年間消費電力（82TWh）に及び，今後も増加し続けると推測されるため，持続可能な開発目標（Sustainable Development Goals; SDGs）に反する重大な問題である [6, 8]．

暗号通貨の維持のために膨大な電力が要求されている問題を解決するため，PoW に代わる様々なコンセンサスプロトコルが考案されてきた．これらのプロトコルは消費される電力を抑えられる一方，完全分散型ではなかったり，プロトコル特有の問題を抱えている場合が多い．PoW はセキュリティにおいて他の代替プロトコルより優れており，依然として既存の暗号通貨の多くは PoW に基づいている．

他にも，消費電力の問題を解決するために，様々な PoW のスキームが考案されている [9–12]．しかし，これらの PoW には社会的な需要の有無や外部システムへの依存など，それぞれに問題点が残されている．柴田氏が考案した PoW では，参加者が自由に最適化問題を設定できるようにしており，分散型のシステムを保ちながら意味のある計算に対して電力を供給している [12]．意味のない計算のために膨大な電力を消費している Hashcash と比較して，環境に良いアルゴリズムであることが分かる．しかし，この論文ではシステムの提案にとどまっており，PoW がセキュリティ上で十分に機能しているかについて議論されていない．

問題点 2：計算資源の無駄遣い

PoW はデータの改ざんを阻止しているが，悪意のある攻撃者がネットワーク全体の過半数の計算力を有している場合，データの改ざんが可能となってしまう．このデータ改ざん方法は，ネットワーク全体の計算力の 51%以上を必要とするため，51%攻撃と呼ばれる．PoW が 51%攻撃を防ぐためには，PoW の計算に特化した専用ハードウェア (ASIC) と通常の CPU の計算能力の差をなくす必要がある [2]．既存の PoW の多くは，メモリハードな問題にすることで ASIC 耐性を実現しているが，計算資源を過度に要求しているため無駄が生じている．

1.3 貢献

本論文では，Hashcash に遺伝的アルゴリズム (Genetic Algorithm; GA) を適用した Meaningful-PoW(mPoW) を提案する．mPoW は，ネットワーク参加者により設定された最適化問題を GA を用いて解く PoW であり，計算資源を過度に要求することなく ASIC 耐性を実現している．

問題点 1 の解決法：GA を用いた mPoW

既存の PoW のほとんどが意味のない計算に対し，膨大な電力を要求しており，環境に奥影響を及ぼしていることが問題視されている．mPoW では，GA の最適化計算を問題としており，他のスキーム同様膨大な計算量を要求する．具体的には，GA の計算により周期的に生成される中間値を Hashcash に入力し，条件を満たすハッシュ値が得られるまで GA の計算を繰り返す問題となっている．Hashcash をベースにしていることで，PoW として必要なセキュリティ上の要件を満たしながら，GA による最適化問題の計算が可能となる．さらに，最適化問題はネットワーク参加者が自由に提示可能な環境にすることで，現在無駄にしている計算力を自由に利用することが可能なシステムとなる．

問題点 2 の解決法：debice-binding 技術を用いた mPoW

既存の PoW の多くは，メモリハードな問題にすることで ASIC 耐性を実現させており，過度な計算資源を要求している．mPoW では，device-binding 技術を適用させることで ASIC 耐性を実現する．Device-binding は，事前に承認された特定のデバイスでのみ PoW の計算を可能にする技術である．初めて PoW を行うハードウェアでは，ソフ

トウェアとハードウェアの間で共通鍵の共有を行う．共通鍵はハードウェア特有の値を用いて生成されるため，ソフトウェアがデバイスを登録したことになる．また，mPoW の計算に共通鍵を用いることで，登録したデバイスで計算が行われているかを監視することが可能となる．mPoW では，ASIC などの独自開発されたハードで行われた計算の結果は，検証時に認められないため，メモリなどの計算資源を過度に要求することなく ASIC 耐性を実現している．

2. Proof of Work

PoW とは，一定の計算量が要求される計算問題であり，ブロックチェーンではコンセンサスアルゴリズムとして利用されている．他にも，ブロックチェーンに保存された過去のデータを改ざんする，51%攻撃を防ぐ機能を果たしている．本章では，PoW の目的や定義，セキュリティ要件，既存の PoW が抱えている問題点について説明する．

2.1 PoW の役割

PoW は，計算に一定の計算量を要する問題のことで，この技術により非中央集権におけるコンセンサスを可能にする [13]．例えば，ビットコインでは PoW として，ハッシュ関数の原像復元問題を利用している．ビットコインの PoW である Hashcash では，ターゲットと呼ばれる基準値があらかじめ定められており，ハッシュ値がターゲット未満となる入力を要求する．これはハッシュ関数の原像を求める問題であり，暗号学的に十分な原像攻撃耐性を有する SHA-2 と RIPEMD をベースにしている Hashcash は，計算量的安全性の保障を可能とする．

証明者が条件を満たす入力を求め，検証者に承認を受けた場合，取引データがブロックチェーンに連結される．参加者はブロックチェーン上のデータを閲覧可能であるが，その改ざんは非常に困難である．複数の取引データからなるブロックは，その名の通りチェーンのように繋がれており，前のブロックの影響を受けている．あるデータを改ざんするためには，そのデータだけでなくそれ以降の全てのブロックを改ざんしなければならない．つまり，それ以降の全ての PoW を攻撃者が解き直さなければならない．しかし，他の証明者は新規ブロックを生成し続けているため，改ざんの完遂は困難となり，暗号通貨の安全性が保障される．具体的には，攻撃者が参加者全体の 51%以上の計算力を有していない場合は改ざんは困難であり，二重払いを防ぐことが可能である．PoW の技術により信頼のおける第三者を介さず，P2P(Peer-to-Peer) ネットワーク上での電子商取引が可能となる．

2.2 PoW の定義

一般的に順方向計算は容易だが，その逆計算は困難となる一方向性関数により PoW は構成される．任意の一方向性関数について，ハードコア述語と呼ばれる予測困難なビット関数の存在が知られており [14]，PoW では次に示す問題 P をハードコア述語として定義する．

$$\mathcal{P} : \mathcal{R} \times \mathcal{I} \times \mathcal{S} \rightarrow \{true, false\}$$

ここでは，$\mathcal{R}$ は問題 $\mathcal{P}$ の難易度を決定するためのパラメータ，$\mathcal{I}$ は $\mathcal{R}$ に準拠する入力，$\mathcal{S}$ は可能解とする．任意の $I \in \mathcal{I}$ 対する問題 $\mathcal{P}_R$ において，$\mathcal{P}(R, I, S) = true$ となる S を見つけ出すアルゴリズム $\mathcal{A}_R$ が存在すると仮定する．

問題 $\mathcal{P}$ を基にした PoW スキームでは，次に示すプロトコルが実行される．

- 証明者はチャレンジの入力 $I \in \mathcal{I}$ とパラメータ $R \in \mathcal{R}$ を公開情報から入手し，$\mathcal{P}(R, I, S) = true$ となる $\mathcal{S}$ を見つけ出して検証者に提示する．この $\mathcal{S}$ を求める作業をマイニングと呼ぶ．
- 検証者は $\mathcal{P}(R, I, S)$ を計算する．

2.3 PoW のセキュリティ要件

PoW はブロックチェーンのコンセンサスアルゴリズムとして用いられているが，同時に 51%攻撃を防ぐなどのセキュリティも担っている．PoW のスキームを開発するにあたり，気をつけなければならない 2 つのセキュリティ要件について言及する．

2.3.1 効率的な検証

PoW では，証明計算で膨大な計算量とメモリを要求しなければならないが，検証計算は低コストで計算可能でなければならない．この要件は *Asymmetry* と呼ばれており，PoW が満たさなければならない要件として Biryukov 氏らが定義している [2]．*Asymmetry* を満たしていない場合，連続して検証を要求する DoS 攻撃が可能となってしまい，ブロックチェーン全体のシステムが停止する恐れがある．

2.3.2 計算の高速化を防ぐ ASIC 耐性

PoW は計算問題を解く競争であるため，ハードウェアの性能差による有利不利が生じる．ASIC などの専用計算機器が開発され，計算の高速化が行われた場合，計算力が独占される可能性が高まる．計算力の独占が実現されると，51%攻撃の危険性につながる．よって，PoW には ASIC による計算の高速化を防ぐ工夫が必要となる．

2.4 既存の PoW の問題点

既存の PoW の問題点として，以下の点が挙げられる．

(1) PoW は膨大な消費電力を無駄遣いしている．
(2) メモリなどのリソースを無駄に要求している．

例えば，Hashcash では「トランザクションデータ」と「直前のブロックのハッシュ値」，「任意に決定したナンス」を SHA-2 や RIPEMD でまとめてハッシュする．そうして得られたハッシュ値が，ターゲットよりも小さければ解

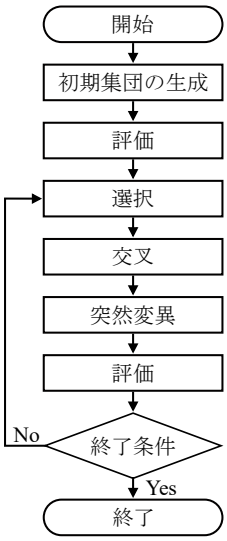

図 1: GA における処理のフローチャート

と祖して認められ，大きければナンスを変更して再度ハッシュしなければならない．これはハッシュ関数の原像復元問題であり，暗号学的に十分な原像攻撃耐性を持っているハッシュ関数がベースであれば，計算量的安全性が保証され，膨大な計算量を必要とする．2021 年 6 月時点では，Hashcash を解くために約 2^{76} 回のハッシュ計算を費やしている [15]．Hashcash が 51%攻撃を防ぐためには，計算量を伴う難解な問題である必要があるが，原像復元問題を解くこと自体に意味があるわけではなく，計算量を確保しつつセキュリティ要件を満たせる計算問題であれば代替可能である [2]．2020 年において，ビットコインの PoW に費やす電力はベルギーの年間消費電力（82TWh）に及び，今後もブロックチェーンの普及に伴い，今後も消費電力は増加すると考えられる [6]．PoW が消費している膨大な電力は，国際的な環境問題であると指摘されている．

また，PoW は ASIC 耐性を持っていなければならない．ASIC のような専用機器を開発・使用されると，証明者が使用している通常のハードウェアとの性能差が生じる．ハードウェアの性能差により，計算力の独占の可能性が高まることで，51%攻撃の危険性が高まる．ASIC は大量のメモリを使用する計算の高速化が困難であるため，メモリハードな PoW にすることで，ASIC 耐性を実現することが可能となる．実際，既存の PoW（Equihash [16] や Merkle Tree Proof [2] など）もメモリハードにすることで ASIC 耐性を実現している．しかし，メモリハードであればあるほど，過剰にハードウェアリソースを要求することになる．

3. 遺伝的アルゴリズム

本研究は柴田氏の先行研究から着想を得て， GA を適用した PoW を考案する [12]．GA は生物の適用進化に関する自然淘汰説に範を得た計算手法であり，様々な最適化問題に適用可能なメタヒューリスティクスとして，回路設計や建物の構造設計，スケジューリングなどで利用されている．本章では，GA の概要や操作について説明する．

3.1 GA の概要

GA は生物が環境に適合して進化していく過程を模倣したメタヒューリスティクスである．GA の研究は 1960 年代後半から 1970 年のはじめに Michigan 大学の Holland 氏らによって始められ，現在では多方面に応用されている [17]．

自然界における生物の進化過程においては，ある世代を形成している個体群，すなわち母集団の中で，より環境に適合した個体が生き残り，次の世代に子孫を残す確率が高くなる．このメカニズムをモデル化し，環境に対して最もよく適合した個体，すなわち目的関数に対して最適値を与えるような解を計算機上で求めようというのが GA の概念である．GA において，個体は設計変数の値がコーディングされた染色体と呼ばれる文字列上で表現され，この染色体をデコーディングすることにより設計変数を読み出し，目的関数の値を計算する．GA はこの母集団に対して選択，交叉，突然変異などの遺伝的操作を繰り返し行うことによって解探索を行う．一般的に，一連の遺伝的操作の繰り返しを世代と呼び，探索が終了するまでに必要となった世代を終了世代数と呼ぶ．図 1 に GA の処理の流れを示す．

3.2 GA の操作

GA の流れを図 1 に示し，それぞれの操作を説明する．

- 母集団の初期化
  あらかじめ設定された数だけランダムに個体を生成する．生成した個体数のことを母集団サイズと呼び，ここで生成した個体の集団を初期母集団とする．
- 評価
  各個体の持つ染色体を問題空間にデコードして個体の評価値を求める．一般に，この評価値を基に個体の適合度値を決定する．適合度値は，個体がその環境に適合している度合いを表す値であり，次世代への生き残りやすさを定量的に示している．そのため，適合度は選択操作の時に用いられる．適合度値が高いほど個体はその環境に適合していると見なす．
- 選択
  生物の適者生存を模倣した処理である．この操作では，生き残る個体を各適合度を基にして定める．
- 交叉
  生物の有性生殖を模倣した処理である．この操作により，個体間で染色体情報が交換される．適合度が高い部分を持った個体同士が交叉すれば，より最適解に近い個体が得られる可能性が高い．母集団の内，何割の個体が交叉するかを交叉率と呼ばれるパラメータによって定める．
- 突然変異

染色体は遺伝子を格納する複数の遺伝子座から構成され，遺伝子座に入りうる遺伝子のことを対立遺伝子という．突然変異とは，染色体上の遺伝子座の遺伝子を別の対立遺伝子に置き換える操作のことであり，自然界における DNA 複写の際に起こるコピーミスを模倣している．各遺伝子座に対して，何割の確率で突然変異が起きるのかを突然変異率と呼ばれるパラメータによって定める．

- 終了判定

  この操作は，あらかじめ定められた終了条件に基づいて GA を終了させるためのものである．

4. mPoW のプロトコル

本論文では，既存の PoW が無駄に電力を消費している問題を解決するため，進化計算である GA を用いた PoW を提案する．このスキームでは，ブロックチェーンの維持に必要となる PoW の計算力をネットワーク参加者に提供し，消費電力の有効活用を可能にする．本章では，GA を適用した PoW の構造案を示す．

4.1 GA のみで構成される PoW の問題

GA のみで構成される PoW について検討する．このスキームは，GA の計算問題であり，いち早く解を得られた証明者に新規ブロックの生成権が与えられる．さらに，問題を提示した参加者からインセンティブを受け取る．

膨大な計算量を必要とする問題へのアプローチは，GA の他にも分散コンピューティングなど様々である．その中でも，GA は目的関数や制約条件，設計変数やその他パラメータを設定することで基本的に同じ流れで計算が可能となる．完全分散型システムにするためには，ネットワーク参加者が自由に問題を提示可能であり，問題ごとに設定を行う管理者が不要であることが必須である．よって，ネットワーク参加者が目的関数やパラメータなどを設定することで，最適化問題の解き方を指定可能な GA は，PoW との親和性が高いと言える．

ブロックチェーンのスケーラビリティを保つために，解が得られるまでの計算時間（以降，ブロックタイムと表記）を一定に維持しなければならない．ビットコインの場合，ブロックタイムが約 10 分間に維持されている [4]．しかし，GA はランダム性に依存しており，GA に要する計算量は推定不可能であるため，GA だけではブロックタイムの維持が困難である．

4.2 解決策：Hashcash によるブロックタイムの維持

GA のみで構成される PoW のスケーラビリティ問題を解決するため，Hashcash のアイデアを利用する．Hashcash は，解の基準となるターゲットを調整することで計算量を操作し，ブロックタイムを維持している [7]．

GA のおおまかな流れは図 1 に示す通り，終了条件を満たすまで，世代毎に同じ処理を反復させる．反復処理特性により，GA では継続的かつ周期的に中間値（各世代の個体群）が生成される．GA では，全ての最適化問題において中間値の形式やサイズが事前に予測可能である．そこで，GA の中間値を Hashcash のナンスとして入力する問題とすることで，ターゲットの調整によりブロックタイムの維持が可能な PoW となる．

GA の問題によっては，計算時間が 10 分に満たない場合や，10 分以上であっても Hashcash の解を得る前に GA の計算が終了する場合が考えられる．いずれの場合もブロックタイムの維持が不可能になるため，Hashcash の計算が終了するまで同じ最適化問題を計算し続ける必要がある．しかし，GA では計算終盤になると解が収束するため，個体に偏りが生じ，同じ個体ばかりを Hashcash に入力し続けることになる．同じ値を入力し続けた場合，永遠に解が得られない．よって，「世代数」も Hashcash に入力させ，常に入力値に変化を持たせる必要がある．

4.3 GA を適用した Hashcash ベースの mPoW のプロトコル

ブロックタイムの維持を可能とする，GA を適用した Hashcash ベースの PoW を提案する．図 2 は，提案する mPoW の構造を示す．本スキームでは，GA により生成された値をナンスとしてみなし，Hashcash に入力する．GA によって生成される値とは，「seed 値」と「世代数」，「各世代で最も適合度が高い個体」の 3 つである．GA の各世代の”評価”が終わる度に，Hashcash のハッシュ計算を行う．「直前のブロックのハッシュ値」と「トランザクションデータ」，「GA により生成された値」のハッシュ値を SHA-3 により求め，ターゲット未満であるか評価する．得られたハッシュ値がターゲット未満である場合，他のネットワーク参加者による検証を開始する．検証では，証明者が行った GA の計算が正当であるか，証明者と同様の計算を検証者たちで分担して行う．

4.4 GA の計算による検証可能性

Hashcash の計算で条件を満たすハッシュ値が得られた場合，他のネットワーク参加者に正当性を検証してもらい，多数決で承認を得られなければならない．検証者は証明者が行った GA の計算を分担して行い，GA の計算により得られた Hashcash の解が正当であるかを検証する．GA の計算に再現性を持たせるために，証明者は GA の計算前に「seed 値」を用いた乱数の固定を行い，検証者に「seed 値」を渡す必要がある．加えて，図 2 のように検証計算の分割を可能にするため，世代ごとの生存個体を一定周期で保存しておく必要がある．GA の計算が進むにつれてメモリを消費していくが，メモリハードと言えるほどメモリは消費

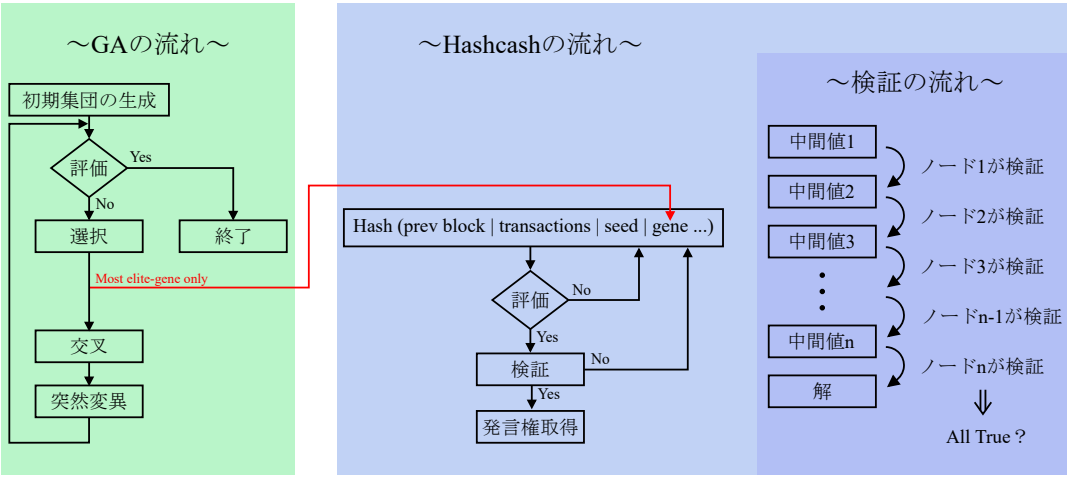

図 2: Meaningful-PoW の構造（GA・Hashcash・検証の流れ）

しない．高羽氏らの研究における，条件を満たすように複数人でスケジュールを組むナーススケジューリング問題を利用した，看護師の勤務表を作成する最適化問題の例では，約 177.5kB のメモリを要求する [18].

4.5 mPoW が満たすセキュリティ要件

PoW が満たさなければならないセキュリティ要件を 2 つ挙げ，mPoW が満たしているかを検証する．

4.5.1 効率的な検証

証明者が計算問題を解いた後，他の参加者による検証が行われる．証明には膨大な計算量やメモリなどのコストを要求しなければならないが，検証には低コストで計算可能でなければならない．

mPoW では，証明に膨大な計算量を要求し，計算により得られる中間値をメモリに保存する．乱数は seed 値で固定し，中間値を保存することで，証明の計算を分担して検証を行うことが可能となる．よって，mPoW の証明には膨大な計算量とある程度のメモリを要求し，検証は低コストで実行可能である．

4.5.2 計算の高速化を防ぐ ASIC 耐性

ASIC などの専用機器を開発・使用されると，証明者が使用しているハードウェア間に性能差が生じ，計算の高速化による 51%攻撃の危険性が高まる．よって，PoW はハードウェアの性能差に影響されにくい，ASIC 耐性を持つ計算問題でなければならない．いくつかの PoW では，ASIC が大量のメモリを使用する計算の高速化が困難であることを利用し，メモリハードな問題にすることで ASIC 耐性を実現している．

mPoW では，計算途中に得られる中間値を保存していくため，計算が進むにつれてメモリを消費していく．しかし，参加者が提示した最適化問題の性質や，PoW の解き方によってはメモリをほとんど使用せずに計算が可能となる．具体的には，GA を最初の数世代だけ計算し，Hashcash の解が得られなければ seed 値を再設定して計算を繰り返すことで，大量のメモリを使用することなく計算が可能となる．よって，mPoW はメモリハードであると断言できず，ASIC 耐性を持っているとは言えない．

4.6 厳密なブロックタイムの維持困難性

mPoW では，Hashcash のターゲットを調整することでブロックタイムを一定に維持することが可能となる．GA の 1 世代あたりの処理時間は最適化問題の難易度により変化するため，ブロックタイムを維持するために問題ごとにターゲットを調整しなければならない．2^n の計算量を要する GA の最適化問題において，1 世代分の実行時間を $time_{GA}$，1 回のハッシュに必要な実行時間を $hash$ とすると，最適化問題に要する実行時間は式 (1) となる．

$$2^n(time_{GA} + hash) \tag{1}$$

しかし，GA の 1 世代あたりの処理時間は理論的に求めることは不可能であるため，厳密にブロックタイムを維持することは困難である．そこで，最適化問題を提示する参加者は事前に数世代分の計算を行い，1 世代あたりにかかる平均実行時間を調べる必要がある．なお，この事前計算で解が得られるような簡単な問題は，インセンティブを払ってまで出題されることを想定していないが，10 分未満で解けた場合は無駄が生じる．

4.7 無駄な計算を省略する方法

GA の計算が 10 分未満で解ける場合，Hashcash の解が見つかるまで計算を続けなければならないため，無駄な計算を繰り返すことになる．そこで，最低計算量の推定方法について検討する．

GA には確率的な収束世代数が存在する [19]．Rylander らの研究では，OneMax 問題における予想収束世代数の推定方法を解明している [20]．OneMax 問題とは，GA にお

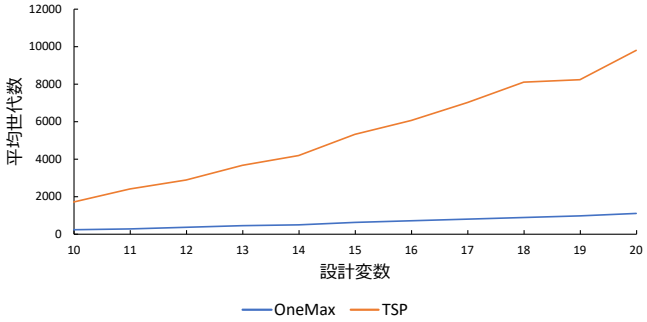

図 3: 設計変数の数による平均計算量の変化 (OneMax vs TSP)

ける遺伝的操作により設計変数を全て 1 にする最適化問題であり，遺伝的アルゴリズムのチュートリアルとして用いられる簡単な例題である．OneMax 問題の予想収束世代数は，遺伝子 x と定数 c を用いて，式 (2) のように表される．

$$c \times log(x) \tag{2}$$

つまり，OneMax 問題の予想収束世代数は遺伝子サイズ $log(x)$ の定数 c 倍となる．

GA を用いた全ての最適化問題の計算において，OneMax 問題が一番簡単であると仮定した場合，mPoW に設定された最適化問題の最低計算量は，OneMax 問題の予想収束世代数以上になると考えられる．実際に，OneMax 問題と巡回セールスマン問題 (Traveling Salesman Problem; TSP) の計算量を実験的に比較した結果，図 3 のように OneMax<TSP となった．TSP とは，複数の都市間の移動において，総移動コストを最小化する最適化問題である．今回の実験では，vcopt [21] というオープンツールを用いて，それぞれ試行回数 1000 回の平均世代数を求めた．

最適化問題と同様のパラメータで設計変数を極端に減らし，計算を行うことで式 (2) の定数 c を求めた後，実際の最適化問題と同様の設計変数の予想収束世代数を求めることで，設定された最適化問題の最低計算量の推定が可能となる．よって，最適化問題の最低計算量と 1 世代あたりの実行時間を式 (1) に代入することで，GA の計算時間が 10 分間以上であるかを推定することが可能となる．

5. ASIC 耐性を持つ mPoW

mPoW では，ネットワーク参加者に膨大な計算力を提供することで，今まで無駄にしてきた電力に意味を持たせることが可能となった．しかし，本スキームは PoW の解き方次第でメモリハードではなくなる可能性があり，ASIC 耐性を持っていない．ASIC などの専用計算機器が開発された場合，実装ハードウェアの性能差が生じてしまい，51% 攻撃の危険性が高まる．

Bock 氏らによる研究では，特定のデバイスでのみ稼働するプログラムの実現方法が示されている [22]．特定のデバイスでのみプログラムを稼働させる技術は，PoW における ASIC 耐性の実現に貢献する．本章では，先行研究の

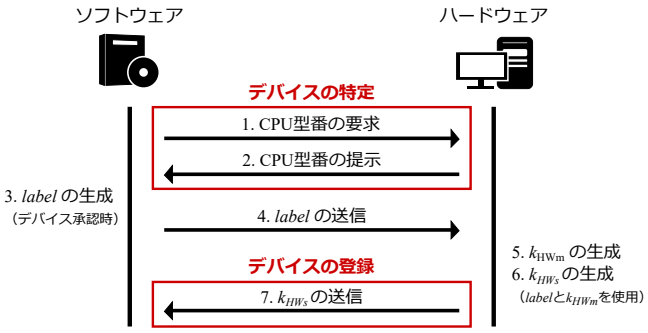

図 4: セットアップ時の device-binding の処理

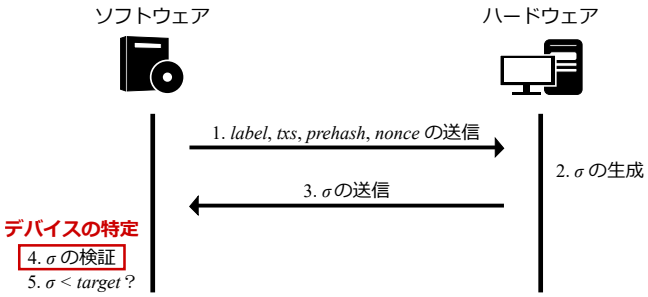

図 5: PoW 時の device-binding の処理

内容を参考にし，PoW に適用可能な device-binding のアイデアを，mPoW を例に提案する．

5.1 特定のデバイスでのみ実行可能にする方法

PoW は特定のデバイスでのみ行われ，ASIC などの一般的でないハードウェア上では計算が不可能な PoW にするために，device-binding の技術を PoW に適用する．

5.1.1 セットアップ

初めて PoW の計算を行うハードウェアの場合，ソフトウェアにハードウェアを登録する．図 4 は，セットアップの処理を示す．まず初めに CPU の型番を要求し，承認されている CPU かどうかを確認する．承認された CPU を使用しているハードウェアであれば，ソフトウェアがハードウェアを識別するための識別子 *label* を型番情報を基に生成し，ハードウェアに送信する．ハードウェアはデバイス特有の秘密鍵 k_{HWm} を生成し，受け取った *label* と k_{HWm} を基に共通鍵 k_{HWs} を生成した後，ソフトウェアと共有する．ソフトウェアとハードウェアで共通鍵を共有することで，デバイスの登録が完了する．

5.1.2 Hashcash の計算による device-binding

mPoW では，先行研究 [22] のアイデアを Hashcash に適用する．図 5 は，device-binding を適用した Hashcash の処理を示す．ソフトウェアがハードウェアに *label* と Hashcash の入力値を渡す．ハードウェアは受け取った *label* と k_{HWm} を用いて k_{HWs} を生成する．ハードウェアは受け取った値 x を，k_{HWs} を鍵として標準暗号アルゴリズム (Advanced Encryption Standard; AES) を用いて暗号化し，得られた暗号文 σ をソフトウェアに送信する．ソフト

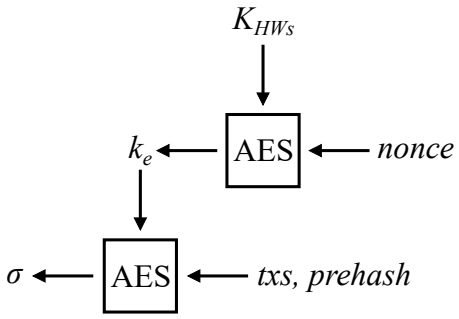

図 6: 検証時に共通鍵 k_{HWs} を秘匿する σ 生成アルゴリズム

ウェア側でも同様の計算を行い σ' を求め，ハードウェアから渡された σ と同値であるかを検証する．$\sigma = \sigma'$ である場合，特定のデバイスで計算されたことが確認されたことになるため，device-binding が機能していることになる．

5.2 秘匿されるべき共通鍵

device-binding を適用した mPoW において，得られた σ がターゲット未満であれば Hashcash の解となる．条件を満たす解 σ が得られた場合，他のネットワーク参加者に検証してもらう必要があり，k_{HWs} を渡さなければならなくなる．そこで，共通鍵を秘匿するために一時的な鍵 k_e を用いる．図 6 は，k_e を用いた σ の生成方法を示す．

k_{HWs} を鍵として，GA の各世代で最も適合度が高い個体を AES を用いて暗号化し，一時的な鍵 k_e を得る．次に k_e を鍵として，トランザクションデータと直前のブロックのハッシュ値を AES を用いて暗号化し，暗号文 σ を生成する．k_e を生成することで，検証の際に渡す値が σ と k_e となり，共通鍵 k_{HWs} を秘匿することが可能となった．

6. 結論

本論文では，参加者が最適化問題を自由に設定可能とする GA を適用した PoW を提案した．その結果，ブロックチェーンの維持に必要となる PoW の計算力をネットワーク参加者に提供可能となり，無駄に消費している電力の有効活用が可能であることを示した．また，mPoW に device-binding 技術を応用することで ASIC 耐性を持たせ，ビットコインなどで使用されている Hashcash が抱える 51%攻撃に対する脆弱性を克服したことを明らかにした．

謝辞

本研究は科研費 挑戦的研究（萌芽）（20K21795）の助成を受けたものです．

参考文献

[1] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[2] Alex Biryukov and Dmitry Khovratovich. Egalitarian computing. Thorsten Holz and Stefan Savage（編）, 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016, pp. 315–326. USENIX Association, 2016.

[3] Alex de Vries. Bitcoin's growing energy problem. *Joule*, Vol. 2, No. 5, pp. 801–805, 2018.

[4] Karl J O'Dwyer and David Malone. Bitcoin mining and its energy footprint. 2014.

[5] Johannes Sedlmeir, Hans Ulrich Buhl, Gilbert Fridgen, and Robert Keller. The energy consumption of blockchain technology: beyond myth. *Business & Information Systems Engineering*, Vol. 62, No. 6, pp. 599–608, 2020.

[6] Alex de Vries. Bitcoin boom: What rising prices mean for the network' s energy consumption. *Joule*, Vol. 5, No. 3, pp. 509–513, 2021.

[7] Adam Back, et al. Hashcash-a denial of service countermeasure. 2002.

[8] Enerdata. グローバルエネルギー統計イヤーブック 2021. https://yearbook.enerdata.jp/electricity/electricity-domestic-consumption-data.html, 2021.

[9] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. *IACR Cryptol. ePrint Arch.*, Vol. 2017, p. 203, 2017.

[10] Sunoo Park, Krzysztof Pietrzak, Joël Alwen, Georg Fuchsbauer, and Peter Gazi. Spacecoin: A cryptocurrency based on proofs of space. *IACR Cryptology ePrint Archive*, Vol. 2015, p. 528, 2015.

[11] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pp. 475–490. IEEE Computer Society, 2014.

[12] Naoki Shibata. Proof-of-search: Combining blockchain consensus formation with solving optimization problems. *IEEE Access*, Vol. 7, pp. 172994–173006, 2019.

[13] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, p. 21260, 2008.

[14] Oded Goldreich and Leonid A Levin. A hard-core predicate for all one-way functions. In *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pp. 25–32, 1989.

[15] COINWARZ. Bitcoin difficulty chart. https://www.coinwarz.com/mining/bitcoin/difficulty-chart, 6 2021.

[16] Alex Biryukov and Dmitry Khovratovich. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Ledger*, Vol. 2, pp. 1–30, 2017.

[17] John H Holland. Genetic algorithms. *Scientific american*, Vol. 267, No. 1, pp. 66–73, 1992.

[18] 高羽実, 吉川大弘, 古橋武. 看護師勤務表作成支援システムのための評価関数の生成手法の開発. 医療情報学, Vol. 28, No. 6, pp. 301–312, 2008.

[19] Carol A Ankenbrandt. An extension to the theory of convergence and a proof of the time complexity of genetic algorithms. In *Foundations of genetic algorithms*, Vol. 1, pp. 53–68. Elsevier, 1991.

[20] Bart Rylander and James Foster. *Computational complexity and the genetic algorithm.* Citeseer, 2001.

[21] vcopt. Pypl. https://pypi.org/project/vcopt/.

[22] Estuardo Alpirez Bock, Chris Brzuska, Marc Fischlin, Christian Janson, and Wil Michiels. Security reductions for white-box key-storage in mobile payments. In *International Conference on the Theory and Application of Cryptology and Information Security*, pp. 221–252. Springer, 2020.