PAPER Special Section on Cryptography and Information Security

mPoW: How to Make Proof of Work Meaningful

Takaki ASANUMA[†a)], ***Nonmember and*** **Takanori ISOBE**[†b)], ***Member***

SUMMARY Proof of Work (PoW), which is a consensus algorithm for blockchain, entails a large number of meaningless hash calculations and wastage of electric power and computational resources. In 2021, it is estimated that the PoW of Bitcoin consumes as much electricity as Pakistan's annual power consumption (91 TWh). This is a serious problem against sustainable development goals. To solve this problem, this study proposes Meaningful-PoW (mPoW), which involves a meaningful calculation, namely the application of a genetic algorithm (GA) to PoW. Specifically, by using the intermediate values that are periodically generated through GA calculations as an 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 a device-binding technology, mPoW can be ASIC resistant without the requirement of a large memory. Thus, we show that mPoW can reduce the excessive consumption of both power and computational resources.

key words: *blockchain, proof of work, energy consumption, genetic algorithm, proof of work*

1. Introduction

1.1 Background

In cryptocurrencies, an important problem is how to prevent double payments. A simple solution is trading through a trusted third-party central authority (such as a bank). However, this solution has the risks of shutting down the entire system owing to an attack on the central server and the risk of misuse by the central authority itself. Therefore, decentralized cryptocurrencies such as Bitcoin [1], use a blockchain system. The basic idea of blockchain is to prevent double payments by maintaining accurate transaction data. Specifically, if the transaction data are stored and if it is impossible to alter the data, then double payments can be prevented.

The Proof of Work (PoW) is the foundational technology used for consensus building in blockchains [2]. Considering the PoW, the majority of participants in the network vote to verify the transaction data. Thus, it is a decentralized system, making it challenging to tamper the data stored in the blockchain. In public blockchains such as Bitcoin, PoW is used to achieve resistance to data tampering attacks.

Manuscript received March 15, 2022.
Manuscript revised August 1, 2022.
Manuscript publicized November 9, 2022.

[†]The authors are with the University of Hyogo, Kobe-shi, 650-0047 Japan.
a) E-mail: takaki.asanuma@gmail.com
b) E-mail: takanori.isobe@ai.u-hyogo.ac.jp

DOI: 10.1587/transfun.2022CIP0010

1.2 Problems of PoW

There are two problems of the current PoW.

Problem 1: Excessive Power Consumption

As a social problem, It has been indicated that PoW wastes a large amount of power [3]–[6]. Hashcash, a PoW applied to Bitcoin, forces the network participants to compute the preimage-finding problem [7]. Because the preimage discovered by Hashcash is not used elsewhere, it consumes power for meaningless calculations. In 2021, it is estimated that the electricity spent on Hashcash will be equivalent to the annual electricity consumption of Pakistan (91 TWh) and it will continue to increase. Therefore, it is a serious problem that goes against the sustainable development goals (SDGs) [6], [8].

Various consensus protocols have been devised as alternatives to PoW to solve the problem of the huge amount of power required to maintain cryptocurrencies. While these protocols consume less power, they are not fully decentralized, or they often have protocol-specific problems. Proof of work is superior to other alternative protocols in terms of security, and many cryptocurrencies are based on PoW.

Various other PoW schemes [9]–[13] have also been devised to solve the power consumption problem. However, each of these PoWs has problems, such as social demand and dependence on external systems. The PoW devised by Shibata allows participants to freely set optimization problems and provide power for meaningful computations while maintaining a decentralized system [12]. Compared to Hashcash, which consumes a large amount of power for meaningless computations, this algorithm is better for the environment. However, this study only proposes a system and does not discuss whether this PoW works well for security.

Problem 2: Excessive Computational Resources

Proof of work prevents data tampering; however, if a malicious attacker has a majority of the computational power in the entire network, then data tampering is possible. This method of data tampering requires more than or equal to 51% of the computational power in the entire network; hence, the attack is known as a 51% attack. Considering a PoW preventing the 51% attack, it is necessary to eliminate the difference in computational power between dedicated hardware

(ASIC) specialized for PoW computation and a normal CPU [2]. Many existing PoWs achieve ASIC resistance by making the problem memory-hard; nevertheless, this is wasteful because it requires excessive computational resources.

1.3 Contribution

In this study, we propose a meaningful-PoW (mPoW) that applies a genetic algorithm (GA) to Hashcash. The mPoW is a PoW that solves an optimization problem set by network participants using GA and is ASIC resistant, without requiring excessive computational resources.

Solution to Problem 1: mPoW with GA

Most existing PoWs [2], [14] require a large amount of power for meaningless computation, which is harmful to the environment. Considering the mPoW, the problem is the optimization of the GA, which requires a large amount of computation, similar to other schemes. Specifically, it is a problem of inputting intermediate values generated by GA computation periodically into Hashcash and continuing the GA computation until a hash value satisfying the condition is obtained. Based on the Hashcash problem, it is possible to compute the optimization problem with the GA while satisfying the security requirements of a PoW. Furthermore, the optimization problem can be freely proposed by the network participants; therefore, the system can freely use the current excessive computational power. Although several PoWs using GA have been devised, no PoW has been proved to meet ASIC resistance[12], [13].

Solution to Problem 2: mPoW using Device-binding

Many of the existing PoWs are ASIC resistant by making them a memory-hard problem, which requires excessive computational resources. Considering mPoW, ASIC resistance is achieved by applying a device-binding technology. Device-binding enables PoW computation only on specific devices that have been approved in advance. Regarding the hardware where PoW is performed for the first time, a symmetric key is shared between the software and hardware. The symmetric key is generated using a hardware-specific value, which indicates that the software has registered the device. In addition, by using the symmetric key for mPoW computation, it is possible to monitor if the computation is being performed on the registered device. In mPoW, the results of the computations performed on dedicated devices (such as ASIC) are not accepted at verification; thereby, achieving ASIC resistance without excessive demand for computational resources such as memory.

1.4 Study Organization

In Sect. 2, we provide an overview of PoW, its security requirements, and the problems of the existing PoWs. In Sect. 3, we present an overview of GA. In Sect. 4, we reveal the flow of mPoW. Section 5 shows how to make the mPoW ASIC resistant, and the conclusion is in Sect. 6.

2. Proof of Work

Proof of work is a computational problem that requires a certain amount of computation and is used as a consensus algorithm in a blockchain. It also prevents 51% attacks that can tamper with past data stored in the blockchain. This section describes the purpose and definition of PoWs, their security requirements, and the problems of the existing PoWs.

2.1 Role of PoWs

A PoW is a problem that requires a certain amount of computation to solve it, and this technology enables a consensus in decentralization [15]. For example, Bitcoin uses the preimage finding problem with the hash function as PoW. Hashcash, a PoW for Bitcoin, has a predefined target value and requires a preimage whose hash value is less than the target. This is the problem of finding a specific preimage of the hash function, and Hashcash, which is based on SHA-2 and RIPEMD with sufficient cryptographic resistance to preimage attacks can guarantee computational security.

The transaction data are linked to the blockchain when the provers request inputs that satisfy the conditions and are approved by the validators. Participants can view the data on the blockchain; however, it is very difficult to tamper with the data. As the name suggests, blocks consisting of multiple transaction data are linked in the form of a chain and are influenced by the previous block. To tamper with the data in a block, that data and all subsequent blocks must be tampered with. This indicates that all subsequent PoW have to be unpacked by the attacker. However, because other provers continue to generate new blocks, it will be difficult for the attacker to complete the tampering. Therefore, the security of the cryptocurrency is guaranteed. Specifically, if the attacker does not have more than or equal to 51% of the total computational power of all the participants, tampering with the data is difficult, and double payments can be prevented. The PoW enables electronic commerce over peer-to-peer (P2P) networks, without the need for a trusted third party.

2.2 Definition of PoW

As defined in [14], PoW has defined the problem

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

as hardcore predicates, where $\mathcal{R}$ is the set of parameters that determine hardness, $\mathcal{I}$ is the set of inputs conforming to $\mathcal{R}$, and $\mathcal{S}$ is the set of possible solutions. We assume that there exists an algorithm (or a family of algorithms) $\mathcal{A}_R$ that solves $\mathcal{P}_R$ on any I, indicating that it finds S such that $\mathcal{P}(R, I, S) = true$.

A PoW scheme based on $\mathcal{P}$ is an interactive protocol

that operates as follows:

1. The Verifier sends a challenge input $I \in \mathcal{I}$ and parameters $R \in \mathcal{R}$ to the prover.
2. The Prover finds solution S such that $\mathcal{P}(R, I, S) = true$ and sends it to the Verifier.
3. The Verifier computes $\mathcal{P}(R, I, S)$.

A non-interactive version (e.g., cryptocurrencies) can be derived easily. Considering this setting, I contains some public values (the last block hash in Bitcoin) and the prover's ID. The prover publishes S to facilitate the verification of the proof by every party.

2.3 Security Requirements for PoW

Proof of work is used as a consensus algorithm for blockchains; nevertheless, it is also responsible for security, such as preventing 51% attacks. To develop a PoW scheme, we discuss two security requirements.

2.3.1 Efficient Verification

Considering PoW, the computation requires a large amount of computation and memory, whereas the validation must be cost effective. This requirement is known as *asymmetry* and is defined by Biryukov et al. as a requirement for the PoW [2]. If *asymmetry* is not satisfied, a DoS attack that requires repeated verification in succession is possible, and the entire blockchain system can be shut down.

2.3.2 ASIC Resistance to Prevent Faster Computation

Because PoW is a race for solving a computational problem, there are advantages and disadvantages owing to differences in hardware performance. If dedicated devices (such as ASIC) are developed to speed up the computation, the possibility of monopolizing the computing power will increase. The monopoly of the computing power leads to the danger of 51% attacks. Therefore, PoW needs to be designed to prevent ASIC from accelerating the computation.

2.4 Problems with the Existing PoWs

The following are the problems with the existing PoWs.

(1) High power consumption.
(2) Excessive memory and other resources.

For example, "transaction data," "hash value of the previous block," and "nonce" are hashed together with SHA-2 in Hashcash. If the output is smaller than the target, it is accepted as the solution, and if it is larger, the nonce must be changed and hashed again. This is the problem of finding a preimage for the hash function. If the hash function has sufficient cryptographic resistance to preimage attacks, it is guaranteed to be secure (considering the computational complexity) and requires a number of computations. In June 2021, it took approximately 2^{76} hash computations to solve Hashcash [16]. To make Hashcash prevent a 51% attack, it must be a difficult problem with a large amount of computation; nevertheless, the preimage-finding problem is not meaningful in itself, and any computational problem that can meet the security requirements while maintaining the amount of computation can be substituted [2]. In 2020, the power consumed by Hashcash was equivalent to the annual power consumption of Belgium (82 TWh) and would continue to increase [6]. The huge amount of power consumed by PoW is indicated as an international environmental problem.

In addition, PoW must be ASIC resistant. If specialized equipment such as ASIC is developed and used, there will be a performance gap with the normal hardware used by the provers, which increases the possibility of the monopolization of the computational power, increasing the risk of a 51% attack. Because ASIC has difficulty in accelerating the computation with a large memory, it is possible to achieve ASIC resistance by making the PoW memory difficult. The existing PoW (such as Equihash [14] and the Merkle Tree Proof [2]) also have difficulty in achieving ASIC resistance. However, the more difficult it is for the memory, the more excessive hardware resources it requires.

3. Genetic Algorithm

In a previous study by Shibata, a PoW applying GA was proposed [12]. However, that paper did not discuss the security requirements that PoWs must meet. Therefore, we propose a GA-based PoW that would satisfy the security requirements based on the previous study. The GA is a computational method inspired by the adaptive evolution of biological systems and is used in circuit design, building structure design scheduling, etc. as meta-heuristics applicable to various optimization problems. This section describes an overview of GA and its operation.

3.1 Overview of GA

A GA is a meta-heuristic that imitates the process by a living organism to adapt to its environment and evolve. The study of GA was initiated by Holland and his colleagues at the University of Michigan in the late 1960s and early 1970s, and has now been applied in many fields [17].

In the evolutionary process in nature, individuals in a population that are better adapted to the environment can survive and have a higher probability of producing offspring in the next generation. The concept of GA is to model this mechanism and to find the individual that best fits the environment (i.e., the solution that gives the optimal value for the objective function). Considering GA, an individual is represented by a string known as a chromosome in which the values of the design variables are coded, and by decoding this chromosome, the design variables are read out, and the values of the objective function are calculated. The GA searches for a solution by repeatedly performing genetic operations such as selection, crossover, and mutation in this

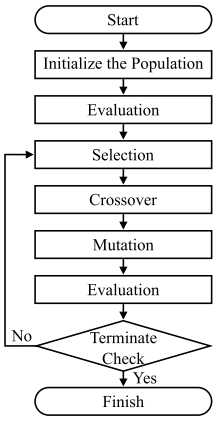

Fig. 1 Flowchart of the process in GA.

population. Generally, the population generated by the genetic operations is counted as a generation, and the number of generations until the end of the search is known as the number of final generations. Figure 1 shows the flow of the GA process.

3.2 Operation of GA

The flow of the GA is shown in Fig. 1 and each operation is explained.

- Initialize the Population
  Randomly generate a predetermined number of individuals.
- Evaluation
  The chromosomes of each individual are decoded to calculate the evaluation value of each individual. Generally, the fitness value of an individual is determined based on this evaluation value. Fitness expresses the degree to which an individual is adapted to its environment and quantitatively indicates how likely it is to survive to the next generation. Therefore, the higher the fitness is, the more adapted the individual is to their environment. Fitness is used in the selection operation.
- Selection
  This process imitates the survival of the fittest in living organisms. In this operation, the surviving individuals are determined based on their fitness.
- Crossover
  This is a process that imitates sexual reproduction in organisms. In this operation, chromosomal information is exchanged between individuals. If individuals with high fitness are crossed with each other, there is a high probability that the individuals closer to the optimal solution will be obtained. The number of individuals to be crossed in a population is determined by a parameter known as the crossover rate.
- Mutation
  A chromosome comprises several loci that contain genes, and a gene that can enter a locus is referred to as an allele. Mutation is the process of replacing a gene at a locus on a chromosome with another allele, imitating the copy errors that occur during a DNA replication in nature. Considering each locus, the probability of mutation was determined by a parameter known as the mutation rate.
- Terminate Check
  This operation is used to terminate the GA based on predetermined termination conditions.

4. Protocol of mPoW

In this study, we propose a PoW with a GA to solve the problem that the existing PoW consumes power unnecessarily. Considering this scheme, the optimization problem using GAs proposed by network participants is the computational problem of the PoW. The power consumption of PoW can be estimated from the computation complexity [18]. Therefore, we can reduce wasted power consumption by changing some of the calculation contents that have been consuming power unnecessarily into optimization problems that are in demand.

4.1 Problem of the PoW Composed of Only GA

We consider a PoW that consists only of GA. This scheme is a GA computational problem, and the provers who are the first to obtain a solution are given the right to generate a new block. In addition, it receives an incentive from the participants who present the problem.

There are various approaches to problems with huge computational requirements, such as distributed computing and GA. Among them, GA can be computed in basically a similar flow by setting the objective function, constraints, design variables, and other parameters. For a fully distributed system, the network participants must present problems freely, and the requirement for an administrator to set up each problem must not exist. Therefore, a GA that allows network participants to specify how to solve an optimization problem by setting the objective function and parameters is highly compatible with PoW.

To maintain the scalability of a blockchain, the computation time until a solution is obtained (known as the block time) must be constant. Regarding Bitcoin, the block time was maintained at approximately 10 min [4]. However, the GA depends on randomness, and the amount of computation required for the GA is impossible to estimate; therefore, it is difficult to maintain the block time with only the GA.

4.2 Solution: Maintaining Block Time with Hashcash

To solve the scalability problem of PoWs consisting of only a GA, we use the idea of Hashcash. Hashcash maintains the block time by controlling the amount of computation by adjusting the target, which is the criterion for the solution [7].

The general flow of the GA is illustrated in Fig. 1, where

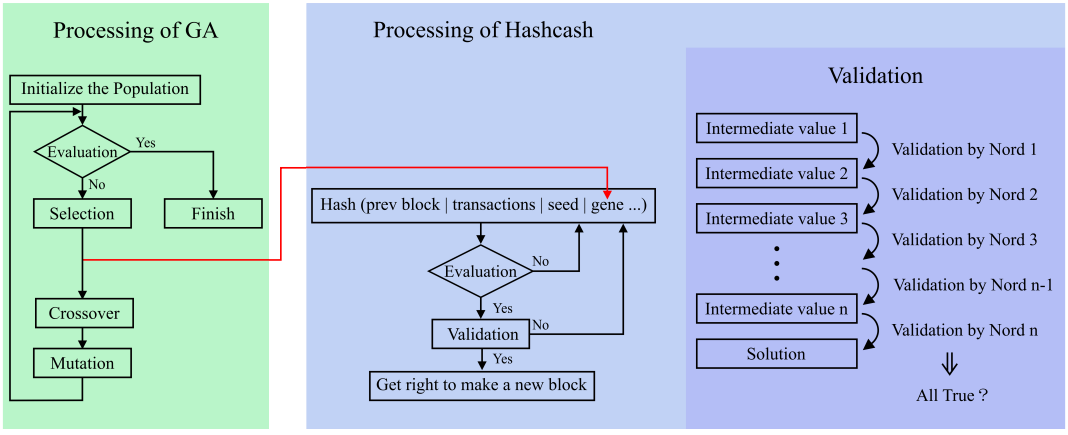

Fig. 2 Processing of the mPoW.

the same process is repeated for each generation until the termination criteria is satisfied. Owing to the iterative property, the GA continuously and periodically generates intermediate values (populations of each generation). Considering the GA, the form and size of the intermediate values can be predicted in advance for all optimization problems. Therefore, using the intermediate values of GA as the input for the hashcash nonce, the problem becomes a PoW that can maintain the block time by adjusting the target.

According as the GA problem, there may be cases where the computation time is less than 10 min or the GA computation terminates before the Hashcash solution is obtained. Regarding either case, the block time cannot be maintained; therefore, the same optimization problem must be continued until the Hashcash calculation is completed. However, considering GA, the individuals converge at the end of the computation. This leads to a bias in the number of individuals, and the same individuals are continuously input to the Hashcash. If we keep inputting the same values, we will not obtain a solution forever. Therefore, it is necessary to input the "number of generations" into the Hashcash to maintain the change in the input values.

4.3 Hashcash-Based mPoW Protocol Applying GA

We propose an mPoW that enables block time maintenance. Figure 2 shows the flow of the mPoW. Considering our scheme, the value generated by the GA is regarded as a nonce and is inputted to Hashcash. The values generated by GA are "seed values," "number of generations," and "individuals with the highest fitness in each generation" After each generation is "evaluated," a hash calculation is performed in Hashcash. The hash value of "the hash value of the previous block", "transaction data," and "the value generated by the GA" is calculated by SHA-3, and it is determined whether the hash value is less than the target. If the output is less than the target, validation by other network participants is initiated. The validator splits the verify into tasks among them and traces the computation of the GA by the prover.

If the problem of GA is not provided, the PoW will be stopped and the blockchain system will be shut down. To avoid this situation, the Hashcash calculation will continue like Bitcoin as long as the GA problem is exhausted.

4.4 Verifiability by the GA Calculation

When a hash value satisfying the condition is found in Hashcash, the hash value must be approved through a majority vote by other network participants for its validity. The validator splits the verify into tasks among them and traces the computation of the GA by the prover. For the GA computation to be reproducible, the provers need to fix the random numbers with "seed" before the GA computation and pass the "seed" to the validators. In addition, it is necessary to store the surviving individuals of each generation at regular intervals to enable the division of the validation calculation as shown in Fig. 2. Although the GA consumes memory as the computation proceeds, it is not memory-hard. Considering the example of the optimization problem to create a nurse's work schedule in the study by Takaba et al., which used the nurse scheduling problem to schedule several people to satisfy a condition, approximately 177.5 kB of the memory was required [19].

4.5 Security Requirements of the mPoW

We list two security requirements of PoW and verify whether mPoW satisfies them.

4.5.1 Efficiency Verification

After the provers have solved the computational problem, other participants perform verification. The proof must require a large amount of computation, memory, and other costs, whereas the validation must be computationally inexpensive.

Considering the mPoW, the proof requires a large amount of computation, and the intermediate values are stored in the memory. By fixing the random number as a seed and storing the intermediate value, the GA computation can be divided and validated. Thus, the proof of the

mPoW requires a large amount of computation and memory, but performing the validation can be inexpensive.

4.5.2 ASIC Resistance to Prevent Faster Computation

When a dedicated device such as ASIC is developed and used, there is a performance difference between the hardware used by the provers, increasing the risk of a 51% attack owing to the faster computation. Therefore, PoW must be ASIC resistant to computational problems that are not affected by hardware performance differences. For some PoWs, it is difficult for ASIC to speed up computations with a large amount of memory; therefore, the ASIC resistance is achieved by making the problem memory difficult.

Considering the mPoW, memory is consumed as the computation proceeds because intermediate values are stored during the computation. However, depending on the property of the optimization problem and solution of the PoW, it is possible to perform the calculation with approximately no memory usage. Specifically, by computing only the first few generations of the GA, if no solution for Hashcash is obtained, the seed is reset, and the calculation is repeated. Using this method, it is possible to perform calculations without using a large amount of memory. Therefore, we cannot conclude that mPoW is memory-hard and ASIC resistant.

4.6 Hardness to Maintain the Block Time

The mPoW can maintain a block time by adjusting the target of the Hashcash. Because the processing time per generation depends on the difficulty of the optimization problem, we adjust the target for each problem to maintain the block time. In the optimization problem of the GA that requires 2^n of computation, if the running time per generation is $time_{GA}$ and $hash$ per hash, the running time required for the optimization problem is given by (1):

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

However, because the processing time per generation cannot be theoretically estimated, it is difficult to maintain the exact block time of the GA. Therefore, the participant presenting the optimization problem needs to perform calculations of several generations in advance to determine the average execution time per generation. Incentives for simple problems that can be solved by this precomputation are not expected. If the problem can be solved in less than 10 min, wastage occurs.

4.7 Eliminating Excessive Calculations

Hashcash requires about 10 minutes of computation to find a solution. If the GA calculation finishes within 10 minutes, Hashcash will not find a solution. In such a case, even after the solution of GA has converged, the calculation must continue until the solution of Hashcash is found. Since the solution of GA has converged, the output will be the same intermediate value. However, since the number of generations is also input to Hashcash, the result of the Hashcash calculation will never be the same. Thus, it is possible to find a solution to Hashcash by continuing the calculation of GA even after the solution of GA has converged. On the other hand, we would like to estimate whether the GA calculation requires more than 10 minutes to avoid as much wasteful computation as possible. Then, we will discuss how to estimate the minimum computation time for the GA.

Such a calculation is considered redundant and wastes power. Therefore, in order to estimate the computation time of GA in advance, we consider the minimum computation time of GA.

There is a stochastic convergence generation number for GA [20]. A study by Rylander et al. shows how to estimate the expected number of convergent generations in the OneMax problem [21]. The OneMax problem is an optimization problem in which all design variables are set to one by genetic operations in GA, and it is a simple example of a problem used as a tutorial for GA. The expected number of generations converged for the OneMax problem is given by (2), where gene x and constant c are used.

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

Alternatively, the number is a constant c multiplied by the gene size $log(x)$.

Assuming the OneMax problem is the easiest of all the optimization problems using GA, the minimum computational complexity of the optimization problem set in mPoW is expected to be more than the expected number of converged generations of the OneMax problem. We experimentally compared the computational complexity of the OneMax problem and the traveling salesman problem (TSP). Figure 3 shows the result of the experiment, and it is found that OneMax<TSP at all times. The TSP is an optimization problem that minimizes the total travel cost for a travel between several cities. Considering this experiment, we used the open tool vcopt [22] to calculate the average generation number for 1000 trials. After finding the constant c in (2) by drastically reducing the design variables with the same parameters in the optimization problem and performing the calculations, we can estimate the minimum computational complexity of the set optimization problem by finding the

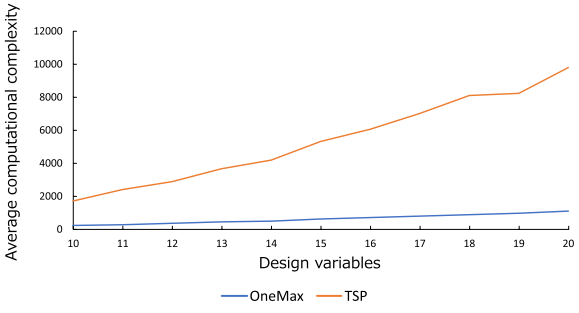

Fig. 3 Change in the average computational complexity depending on the number of design variables (OneMax vs TSP).

expected number of converged generations in the design variables similar to the actual optimization problem. Therefore, by substituting the minimum computational complexity of the optimization problem and running time per generation into (1), we can determine whether the computational complexity of the GA is more than 10 min.

5. ASIC Resistant mPoW

The mPoW can provide enormous computational power to the network participants. However, this scheme may not be memory-hard (depending on how to solve the PoW), and it is not ASIC resistant. If a dedicated device such as an ASIC is developed, the performance gap of the hardware will increase the risk of a 51% attack.

The study by Bock et al. shows how to make a program run only on a specific device [23]. The technique of running a program only on a specific device contributes to achieving ASIC resistance in PoW. In this section, we propose the idea of device-binding applicable to PoW.

5.1 Making it Possible to Run Only on a Specific Device

We apply device-binding technology to mPoW to make the PoW run only on a specific device and incomputable on uncommon hardware such as ASIC. We assume that the PoW software has not been spoofed or tampered with and can be trusted in this paper.

5.1.1 Setup

Using new hardware to compute PoW for the first time, the software registers the hardware. Figure 4 shows the setup process. First, it requests the model number of the CPU and checks whether it is an approved CPU. If the hardware uses an approved CPU, the software generates an identifier *label* based on the model number to identify the hardware and sends it to the hardware. The hardware generates a device-specific secret key k_{HWm} (hardware main key; HWm), generates a symmetric key k_{HWs} based on the received *label* and k_{HWm}, and shares it with the software. The registration of the device is completed when the software and hardware share the symmetric key.

5.1.2 Device-Binding with Hashcash

The mPoW applies the ideas of a previous study [23] to Hashcash. Figure 5 shows the process of Hashcash applying device-binding. The software passes *label* and the input value of the hashcash to the hardware. The hardware generates k_{HWs} with the received *label* and k_{HWm}. The hardware encrypts the received value x, using the advanced encryption standard (AES) algorithm with k_{HWs} and sends the ciphertext σ to the software. The software performs the same calculation to find σ' and validates that it is the same value as σ passed from the hardware. If $\sigma = \sigma'$, then it is confirmed that the calculation is performed on a specific device.

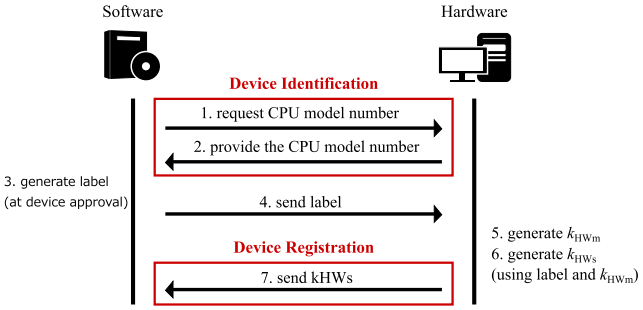

Fig. 4 Processing device-binding during the setup.

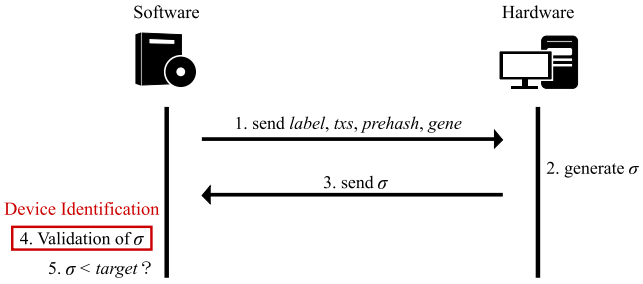

Fig. 5 Processing device-binding during PoW.

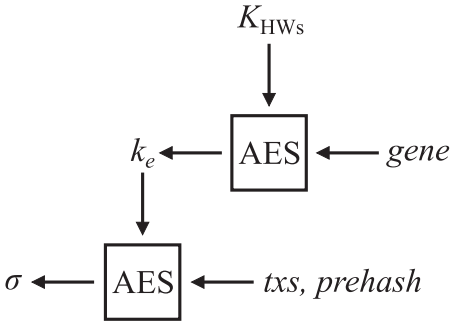

Fig. 6 Algorithm for generating σ to maintain the secrecy of the symmetric key k_{HWs} during validation.

Therefore, it indicates that device-binding is working.

5.2 Keeping the Symmetric Key as a Secret

The mPoW with device-binding accepts σ as a Hashcash solution if the value is less than the target. If σ is found to satisfy the condition, it needs to be validated by other network participants, and k_{HWs} must be passed. Therefore, an ephemeral key k_e is used to keep the symmetric key as a secret. Figure 6 shows how to generate σ with k_e.

We encrypt the individuals with the highest fitness in each generation of the GA using AES with k_{HWs} and obtain k_e. Subsequently, the transaction data and hash value of the previous block are encrypted using AES with k_e, and the ciphertext σ is generated. Although it is necessary to pass σ and k_e to the validators, it is now possible to maintain the symmetric key k_{HWs} a secret.

6. Conclusion

In this study, we proposed the new PoW applying a GA that allows participants to freely set the optimization problem. Thus, we showed that the computational power required to maintain the blockchain can be provided to the network participants, and it is possible to effectively utilize the wasted power consumption. In addition, by applying the device-binding technology to mPoW, we made it ASIC resistant and overcame the vulnerability of Hashcash to a 51% attack. In order to make mPoW more practical, we think it is necessary to implement this scheme and also evaluate its performance.

Acknowledgments

Takanori Isobe is supported by Grants-in-Aid for Scientific Research (Houga) (KAKENHI 20K21795) for the Japan Society for the Promotion of Science.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system, Technical report, Manubot, 2019.
[2] A. Biryukov and D. Khovratovich, "Egalitarian computing," 25th {USENIX} Security Symposium ({USENIX} Security 16), pp.315–326, 2016.
[3] A. de Vries, "Bitcoin's growing energy problem," Joule, vol.2, no.5, pp.801–805, 2018.
[4] K.J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," 2014.
[5] J. Sedlmeir, H.U. Buhl, G. Fridgen, and R. Keller, "The energy consumption of blockchain technology: Beyond myth, Bus. Inf. Syst. Eng., vol.62, no.6, pp.599–608, 2020.
[6] A. de Vries, "Bitcoin boom: What rising prices mean for the network's energy consumption," Joule, vol.5, no.3, pp.509–513, 2021.
[7] A. Back, "Hashcash - A denial of service counter-measure," 2002.
[8] Decrypt, "Bitcoin energy consumption this year already more than all of 2020," https://decrypt.co/80897/bitcoin-energy-consumption-year-already-more-2020, 2021.
[9] M. Ball, A. Rosen, M. Sabin, and P.N. Vasudevan, "Proofs of useful work," IACR Cryptol. ePrint Arch., 2017:203, 2017.
[10] S. Park, K. Pietrzak, J. Alwen, G. Fuchsbauer, and P. Gazi, "Spacecoin: A cryptocurrency based on proofs of space," IACR Cryptology ePrint Archive, 2015:528, 2015.
[11] A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz, "Permacoin: Repurposing bitcoin work for data preservation," 2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18–21, 2014, pp.475–490, IEEE Computer Society, 2014.
[12] N. Shibata, "Proof-of-search: Combining blockchain consensus formation with solving optimization problems," IEEE Access, vol.7, pp.172994–173006, 2019.
[13] F. Bizzaro, M. Conti, and M.S. Pini, "Proof of evolution: Leveraging blockchain mining for a cooperative execution of genetic algorithms," 2020 IEEE International Conference on Blockchain (Blockchain), pp.450–455, IEEE, 2020.
[14] A. Biryukov and D. Khovratovich, "Equihash: Asymmetric proof-of-work based on the generalized birthday problem," Ledger, vol.2, pp.1–30, 2017.
[15] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized Business Review, p.21260, 2008.
[16] COINWARZ, Bitcoin difficulty chart, https://www.coinwarz.com/mining/bitcoin/difficulty-chart, June 2021.
[17] J.H. Holland, "Genetic algorithms," Sci. Am., vol.267, no.1, pp.66–73, 1992.
[18] M. Platt, J. Scdlmeir, D. Platt, J. Xu, P. Tasca, N. Vadgama, and J.I. Ibañez, "The energy footprint of blockchain consensus mechanisms beyond proof-of-work," 2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C), pp.1135–1144, IEEE, 2021.
[19] M. Takaba, T. Yoshikawa, and T. Kodama, "Development of a generation method of evaluation functions for the nurse scheduling support system," Department of Medical Informatics, vol.28, no.6, pp.301–312, 2008.
[20] C.A. Ankenbrandt, "An extension to the theory of convergence and a proof of the time complexity of genetic algorithms," Foundations of Genetic Algorithms, vol.1, pp.53–68, Elsevier, 1991.
[21] B. Rylander and J. Foster, Computational Complexity and the Genetic Algorithm, Citeseer, 2001.
[22] vcopt, Pypl. https://pypi.org/project/vcopt/
[23] E.A. Bock, C. Brzuska, M. Fischlin, C. Janson, and W. Michiels, "Security reductions for white-box key-storage in mobile payments," International Conference on the Theory and Application of Cryptology and Information Security, pp.221–252, Springer, 2020.

Takaki Asanuma was born in 1996. He received his B.E. degree from Doshisha University, Japan, in 2020. He is currently a master's student at University of Hyogo, Japan. His research interest is blockchain.

Takanori Isobe received the B.E., M.E., and Ph.D. degrees from Kobe University, Japan, in 2006, 2008, and 2013, respectively. From 2008 to 2017, he worked at the Sony Corporation. Since 2017, he has been an Associate Professor at University of Hyogo. His current research interests include information security and cryptography.