A Proof of Work Based on Key Recovery Problem of Cascade Block Ciphers with ASIC Resistance

Takaki ASANUMA (a), Nonmember and Takanori ISOBE (b), Member

SUMMARY Hashcash, which is a Proof of Work (PoW) of bitcoin, is based on a preimage problem of hash functions of SHA-2 and RIPEMD. As these hash functions employ the Merkle-Damgard (MD) construction, a preimage can be found with negligible memory. Since such calculations can be accelerated by dedicated ASICs, it has a potential risk of a so-called 51% attack. To address this issue, we propose a new PoW scheme based on the key recovery problem of cascade block ciphers. By choosing the appropriate parameters, e.g., block sizes and key sizes of underlying block ciphers, we can make this problem a memory-hard problem such that it requires a lot of memory to efficiently solve it. Besides, we can independently adjust the required time complexity and memory complexity, according to requirements by target applications and progress of computational power. key words: blockchain, Proof of Work, cascade block ciphers, key recovery problem

1. Introduction

1.1 Background

The essential issue for cryptocurrencies is to prevent duplicated payments. A simple solution is to trade through a trusted third-party central institution such as a bank. However, this solution potentially has a risk such that the entire system may be shut down because an attack on the central server or the central authority itself may do the malicious behavior. To mitigate these risks, cryptocurrencies with a decentralized system, e.g., Bitcoin and Ethereum, are based on blockchain. The basic idea behind blockchain is to consistently keep the correct transaction data. Since the transaction data is stored in a form that everyone can see and verify its correctness as a distributed ledger, malicious users cannot manipulate the data. To maintain this property, the blockchain needs to be secure against so-called 51% attacks in which malicious users monopolize more than half of the computational resources.

Proof of Work (PoW) is a consensus algorithm in the blockchain [1]. To prevent 51% attacks, PoW should minimize the difference of computational advantage between dedicated hardware (ASIC) and ordinary CPUs as possible. A solution for this purpose is a memory-hard function that consumes a large number of memory accesses [2]–[4]. Since memory-consuming computations cannot be sufficiently speeded up by ASIC, an advantage for mining pools equipping dedicated ASIC over ordinary CPUs is limited. In particular, the property of steep time-memory tradeoff such that the amount of computational complexity exponentially increases when the amount of memory is reduced is very important for ASIC resistance [1]. Biryukov and Khovratovich propose the seven security requirements for the problem of PoW, including steep time-memory tradeoff [1].

Hashcash, which is a PoW of bitcoin, is based on a preimage problem of hash functions of SHA-2 and RIPEMD [5]. Since these hash functions employ the Merkle-Damgard (MD) construction, a preimage can be found with negligible memory by the brute force search. Due to the lack of the property of steep time-memory tradeoff, Hashcash has a potential risk of 51% attack by mining pools equipped with ASIC.

As an alternative PoW for Hashcash, Equihash [6] and Merkle Tree Proof (MTP) [1] are proposed. These PoWs were developed to solve the problems of Hashcash, however, they do not completely meet all seven security requirements defined by Biryukov and Khovratovich [1].

Equihash is a PoW that is used in Zcash and others [6]. Although Equihash has ASIC resistance because it requires a large amount of memory, it cannot independently adjust the required time and memory complexity. MTP is a PoW used in Zcoin [1]. MTP also achieves ASIC resistance similar to Equihash, however, it was pointed out that a more efficient algorithm with smaller memory is available [7]. It means that these do not satisfy properties of flexibility and optimization-free, respectively, which are the properties required by Biryukov and Khovratovich [1].

1.2 Our Contribution

In this paper, we aim to develop a new PoW that adequately satisfies steep time-memory tradeoff and other security requirements, especially for flexibility and optimization-free. To achieve this goal, we utilize a key recovery problem of 2-round cascade block ciphers (2CBC) as an underlying problem instead of the preimage problem of hash functions for hashcash. Specifically, in our PoW, the verifier sends a pair of plaintext/ciphertext to the prover, then the prover finds the corresponding key of 2CBC that maps the given plaintext to the given ciphertext.

In the following, we summarize how to achieve steep time-memory tradeoff, flexibility, and optimization-free in...
key recovery problems of 2CBC.

**Steep time-memory tradeoff:** For solving the key recovery problem of 2CBC efficiently, there are two approaches: a brute force search and a meet-in-the-middle (MITM) approach. Although the brute force search is feasible with negligible memory, the MITM approach requires a large amount of memory to store intermediate values. Our idea for realizing a memory-consuming PoW is to make the MITM much more time-efficient than the brute force search by properly choose parameters of 2CBC. In this case, mining pools cannot avoid choosing the memory-consuming MITM approach as an efficient algorithm of PoW.

**Flexibility:** For the problem of the key recovery problem on the 2CBC, by fixing some values of keys and internal states to predetermined values, we can flexibly control the time and memory complexity to efficiently solve this problem, according to requirements. In other words, required time and memory complexity are independently tunable.

**Optimization-free:** To avoid a clever prover getting the advantage over the others the advertised algorithm must be the most efficient algorithm to date. Indeed, several variants of the first version of MTP [1] were broken [7]. Key recovery problems of 2CBC have been significantly evaluated by symmetric-key cryptography communities over the past few decades. Since these problems have never been broken before, we can have a high confidence in the difficulty of breaking that. It is very important from the point of the optimization-free.

Therefore, our PoW schemes satisfy steep time-memory tradeoff, flexibility, and optimization-free. As far as we know, our schemes first achieve all of these three important properties.

1.3 Paper Organization

In Sect. 2, we describe an overview of PoW, its security requirements, and the problems of existing PoWs. In Sect. 3, we explain the details of 2CBC and its key recovery problems. In Sect. 4, we reveal the parameters for the 2CBC for realizing the ASIC resistance. Section 5 gives instantiations of our schemes by properly adjusting the parameters. Section 6 shows how to achieve flexibility and practical parameters. Section 7 discusses the other security requirements and compares them with other candidates, and the conclusion in Sect. 7.

2. Proof of Work

This section explains the purpose, definition, and security requirements of PoW and then discusses the problem of existing hash-based schemes.

2.1 The Purpose of PoW

PoW is a problem that requires a certain amount of time to solve and is used as a consensus algorithm for decentralization. For example, Bitcoin utilizes a preimage problem of hash functions called Hashcash as an underlying PoW. Specifically, miners/prover should find a preimage of a digest value of hash function whose first $d$-bit are all zero as a solution for PoW. This is the preimage problem of hash functions of SHA-2 or RIPEMD.

In the blockchain, after the prover solves the PoW, the transaction data is connected to the blockchain. Entities can see the data on the blockchain, however, it is very difficult to tamper with it, because blocks consisting of multiple transaction data are affected by the previous block. To modify a piece of data, the attacker must modify not only the piece of data but also all subsequent blocks. In other words, the attacker needs to solve all corresponding PoW.

If the miner does not have more than 51% of all computing resources, such computation is infeasible in principle. Thus, the blockchain using PoW realizes electronic commerce on peer-to-peer (P2P) networks without the need for trusted third parties.

2.2 Definition of PoW

As defined in [6], PoW has defined a problem

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

as hardcore predicates, where $\mathcal{R}$ is the set of parameters that determine the 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 is an algorithm (or a family of algorithms) $\mathcal{A}_R$ that solves $\mathcal{P}_R$ on any $I$, i.e. finds $S$ such that $\mathcal{P}(R, I, S) = \text{true}$.

A PoW scheme based on $\mathcal{P}$ is an interactive protocol, which 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 $\mathcal{P}(R, I, S) = \text{true}$ and sends it to the Verifier.
3. The Verifier computes $\mathcal{P}(R, I, S)$.

A non-interactive version (e.g., for cryptocurrencies) can be derived easily. In this setting, $I$ contains some public value (last block hash in Bitcoin) and prover’s ID. The prover publishes $S$ so that every party can verify the proof.

2.3 Security Requirements for PoW

Biryukov and Khovratovich propose the seven security requirements of problem $\mathcal{P}$ and algorithm $\mathcal{A}$ [1].

1. **Amortization-free:** Producing $q$ outputs should be $q$ times as expensive.
2. **Asymmetry:** The solution must be short enough and verified quickly using little memory to prevent DoS attacks on the verifier.
3. **Steep Time-Memory Tradeoff:** The time-space trade-offs must satisfy the steep relationship to prevent any price-performance reduction.
4. **Flexibility**: The time and memory complexity must be tunable independently to sustain a constant mining rate.

5. **Optimization-free**: To avoid a clever prover getting the advantage over the others the advertised algorithm must be the most efficient algorithm to date.

6. **Progress-free**: The algorithm must be progress-free to prevent centralization: the mining process must be stochastic so that the probability to find a solution increases steadily with time and is also non-zero for the early stage.

7. **Parallel-unfriendly**: Parallelized implementations must be limited by the memory bandwidth.

Among these requirements, (3) is the most important for the ASIC resistance as it requires a large amount of memory and random memory access.

### 2.4 Problems of Hashcash

Hashcash, which is a PoW of the bitcoin, is based on a preimage problem of hash functions of SHA-2 and RIPEMD. Since these hash functions employ the Merkle-Damgard (MD) construction, a preimage can be found with negligible memory by the brute force search. Due to the lack of the property of steep time-memory tradeoff, Hashcash does not have resistance against the so-called 51% attack by mining pools equipping ASIC.

As an alternative PoW for Hashcash, Equihash [6] and Merkle Tree Proof (MTP) [1] are proposed. These PoWs were developed to solve the problems of Hashcash, however, they do not completely meet all seven security requirements defined by Biryukov and Khovratovich [1].

Equihash is a PoW that is used in Zcash and others [6]. Although Equihash has ASIC resistance because it requires a large amount of memory, it cannot independently adjust the required time and memory complexity. MTP is a PoW used in Zcoin [1]. MTP also achieves ASIC resistance similar to Equihash, however, it was pointed out that a more efficient algorithm with smaller memory is available [7]. So it is not clear whether it can sufficiently meet optimization-free [7]. It means that these do not satisfy properties of flexibility and optimization-free, respectively, which are the required properties by Biryukov and Khovratovich [1].

### 3. Our Approach

In this paper, we aim to develop a new PoW that adequately satisfies steep time-memory tradeoff and other security requirements, especially for flexibility and optimization-free. To achieve this goal, we utilize a key recovery problem of 2-round cascade block ciphers (2CBC) instead of the preimage problem of hash functions.

2CBC consist of n-bit two block ciphers, e.g. AES, which takes k1-bit key $K_1$ and k2-bit key $K_2$ as inputs, respectively [8]. An n-bit plaintext $M$ is encrypted twice by n-bit block ciphers to obtain an n-bit ciphertext $C$ as shown in Fig. 1.

![Fig. 1 The details of 2CBC](image)

#### 3.1 Overview of Our PoW

In our PoW, the verifier sends a pair of plaintext/ciphertext to the prover, then the prover finds the corresponding key $K_1$ and $K_2$ of 2CBC that maps the given plaintext to the given ciphertext. Specifically, the new PoW scheme based on $P$ is a non-interactive protocol, which operates as follows:

1. The server sends a challenge input $M$ and $C$, and parameters (e.g., $k_1$ and $k_2$) to the Prover.
2. Prover finds keys $K_1$ and $K_2$ such that the relationship between the given plaintext and ciphertext is valid, and sends them to the Verifier.
3. Verifier computes $enc(M)$ with $K_1$ and $K_2$.

The general flow of the process is the same as for other PoWs [1], [7]. First, the prover receives the plaintext, the ciphertext, and parameters. Then, the prover searches for keys that satisfy these conditions. The plaintext $M$ and the ciphertext $C$ are generated based on the transaction data of about 8000000-bit. The transaction data is converted to an $n$-bit hash value using SHA-3, and we consider this hash value as $M$. The plaintext $M$ is again hashed to generate the $n$-bit ciphertext $C$. After they find keys, he publishes them. The verifier checks whether the keys are appropriate. If they are correct, authorizes the transaction data to be connected to the blockchain.

#### 3.2 Key Recovery Problem of 2CBC

There are two approaches for key recovery attacks on the 2CBC such that the brute force attack and Meet-in-the-Middle attack (MITM).

##### 3.2.1 Brute Force

A brute force attack first guesses values of $K_1$ and $K_2$ and check whether it satisfies the $n$-bit relationship between plaintext $M$ and ciphertext $C$ that are sent from the verifier. The time complexity is given as follows:

$$\text{Time} : 2^n$$

(1)

The brute force attack is feasible with negligible memory.

##### 3.2.2 MITM

In MITM, the adversary finds a collision in total of $2^n$ intermediate values which are computed by forward computation $f(x)$ with $K_1$ and backward computation $g(x)$ with $K_2$, respectively.
respectively, as shown in Fig. 2. The number of intermediate values to be collected follows a birthday problem, i.e. a total of \(2^n\) intermediate values are needed.

Depending on the size of \(n, k_1, k_2\), the time and memory complexity varies. Specifically, it follows the trade of \(TM = N\), where \(T\) is time complexity and \(M\) is memory requirement, and \(N = 2^n\). The minimum values of time and memory complexity are given as follows.

- Time: \(2^{n/2}\)
- Memory: \(2^{n/2}\)

Note that if the length of the key is too small, MITM is impossible because we can not collect the \(2^n\) intermediate values.

### 3.3 Floyd Circle-Finding Algorithm

Floyd’s algorithm finds a collision between two functions by interleaving the calls to \(f(x)\) and \(g(x)\) during the detection of the cycle [9]. Floyd’s algorithm is based on a selection function that evaluates either \(f(x)\) or \(g(x)\) with equal probability. Thus, a collision is an actual one for \(f(x)\) and \(g(x)\) with a probability of 1/2. Consequently, the search has to be repeated twice. Using Floyd’s cycle-finding algorithm for MITM [10], the time complexity is almost \(2^{n/2}\) with negligible memory.

In this paper, we call this MITM “F-MITM” and the standard MITM “S-MITM”.

### 3.4 Hellman’s Time-Memory Tradeoff

The Hellman’s time-memory tradeoff has the potential to reduce memory requirements [11]. While S-MITM stores all intermediate values of \(2^{n/2}\) in memory, Hellman’s approach stores only a part of intermediate values. A ciphertext created from a plaintext is continuously input as the next plaintext, and a chain of plaintext and ciphertext is created. The values of the chain are stored in Hellman’s table at regular intervals of \(l\). At the creation of the table, the time complexity is equal to that of S-MITM \((= 2^{n/2})\) [12]. Hellman’s online computation is performed similar to S-MITM’s that of \(2^{n/2}\). In this case, Hellman’s table stores only some of the values, so it requires \(l\) times more calculations \((= 2^{n/2} \times l)\) for validation than S-MITM. Thus, the Hellman’s approach can reduce the memory consumption, however the computational complexity increases. The cost of key search with Hellman’s time-memory tradeoff is as follows.

- Time: \(2^{n/2} \times l\)
- Memory: \(\frac{2^{n/2}}{l}\)

The time and memory complexity is always \(TM = N\), even when we use Hellman’s table. Since PoW is a computational race and miners want to use as little time complexity as possible, Hellman’s time-memory tradeoff can not improve S-MITM.

### 4. Finding Parameters for PoW Based 2CBC

This section explores parameters of the 2CBC so that the key recovery problem of 2CBC requires a large amount of memory to efficiently solve, i.e. the memory-efficient approach is much less time-efficient than the one using a large amount of memory. In this case, mining pools cannot avoid choosing the memory-consuming MITM approach as an efficient algorithm of PoW.

#### 4.1 Criteria for Parameter Selection

Our first goal is to develop a PoW that has the property of steep time-memory tradeoff for the ASIC resistance. To achieve this purpose, we should properly choose the parameters of 2CBC such that S-MITM, which is the most memory-consuming approach, is much more time-efficient than others in any time-memory tradeoff point of \(MT = 2^n\). Specifically, we will uncover the parameters of the 2CBC that satisfy the following criteria.

- (A) S-MITM can find a solution with high probability.
- (B) S-MITM is much more time-efficient than brute force.
- (C) S-MITM is much more time-efficient than F-MITM.

#### 4.2 A Parameter for (A)

We must collect a total of \(2^n\) the intermediate values which are computed by \(f(x)\) and \(g(x)\). If the key lengths \(k_1\) and \(k_2\) are too small for the block length \(n\), it is impossible to obtain a sufficient solution. Assume an example as in Fig. 3. In this example, it is possible to collect up to \(2^{80}\) solutions from \(f(x)\) and \(2^{40}\) solutions from \(g(x)\). To use MITM, we need to collect a total of \(2^n\) solutions, i.e. \(2^{128}\) solutions. Thus, it can not find a solution with high probability.

Since intermediate value obtained from 2CBC is \(2^{k_1+k_2}\),
it must satisfy the following conditions.

\[ n \leq k_1 + k_2 \quad (2) \]

4.3 A Parameter for (B)

Time complexity for a brute force attack is estimated as Eq. (1). Since the maximum of time complexity required by S-MITM is \(2^n\), the time complexity of S-MITM is significantly lower than that of the brute force attack if Eq. (2) holds.

4.4 A Parameter for (C)

By using Floyd’s algorithm [9], S-MITM may be feasible with a time complexity of \(2^{n/2}\) and negligible memory. To utilize Floyd’s algorithm, the input and output lengths of \(f(x)\) and \(g(x)\) need to be the same difference. In our scheme, the input length is \(k_i\)-bit and the output length is \(n\)-bit. Here, we consider whether an F-MITM is feasible in two cases of \(k_i > n\) and \(k_i < n\).

4.4.1 For \(k_i > n\)

If the input length \(k_i\) is larger than the output length \(n\), the remaining \((k_i - n)\) bit should be fulfilled for the next input, as shown in Fig. 4. This problem can be easily solved by zero-padding. Thus, F-MITM is feasible with the same time complexity even if \(k > n\).

4.4.2 For \(k_i < n\)

If the input length \(k\) is less than the output length \(n\), the surplus of \((n - k)\)-bit should be removed for the next input as shown in Fig. 5. In that case, even if an adversary can find a collision in \(k\)-bit intermediate values, we should check whether the remaining \((n - k)\) bit are also the same. Thus, F-MITM requires the time complexity of \(2^{k/2} \times 2^{n-k}\).

The time complexity of the F-MITM must be greater than the maximum time complexity of the S-MITM. The condition is expressed in the following equation.

\[ 2^k < 2^{k/2} \times 2^{n-k} \]

\[ \Rightarrow k < n - \frac{k}{2} \]

We choose parameters that satisfy the following conditions to make F-MITM match inefficient compared to S-MITM.

\[ k_1 < \frac{2}{3}n \quad (3) \]

\[ k_2 < \frac{2}{3}n \quad (4) \]

4.5 Summary of Parameter Selection

In this section, we listed the conditions that the parameters of the 2CBC must satisfy and examined each of them. To summarize Eq. (2)-(4), the parameters must satisfy the following equations.

\[ k_1 < \frac{2}{3}n \quad (5) \]

\[ k_2 < \frac{2}{3}n \quad (6) \]

\[ n \leq k_1 + k_2 \quad (7) \]

5. Instantiations

In this section, we propose a PoW based on variants of 2CBC that achieves *steep time-memory tradeoff*. Also, we
explain how to uniquely define the time and memory complexity required by S-MITM.

For $n = 128, 256, 512$, we give three instances that satisfy Eq. (5)-(7). Figure 6 shows the time and memory complexity of S-MITM, F-MITM, and brute force attack for each instance. F-MITM and brute force attack require negligible memory, but require more time complexity than S-MITM. Since PoW is a computational race, we expect miners to choose S-MITM. We also find that the more memory is consumed in S-MITM, the smaller the time complexity becomes. Since the prover has an advantage in using more memory, our PoW has an ASIC resistant.

6. Flexibility

We explain how to independently control the time and memory complexity in our PoW.

6.1 Tuning Memory Complexity

To control the required memory complexity, we adjust the key size $k_2$ by fixing the first $t$-bit of $K_2$ with zero. Figure 7 shows the change of memory complexity by adjusting $k_2$ when $n = 256$ and $k = 160$. The smaller $k_2$ is, the smaller the upper limit of time complexity becomes. As the upper limit of time complexity decreases, more memory complexity is required. We can lower the lower limit of memory complexity by reducing $k_2$ with the $r$-bit limit. For example, when $k_2 = 160$, the range of memory complexity is from $2^{96}$ to $2^{160}$ in Fig. 7. If $t$ increases to decrease $k_2$, e.g., $t = 64$, the lower limit of the memory lower to $2^{96}$. We must be careful not to break Eq. (5)-(7) by excessively reducing $k_2$. If the condition is broken (in this case $t > 64$), MITM is disabled. In such a case, we can use only brute force attack.

6.2 Tuning Time Complexity

To increase the time complexity, we set a limit that the first $r$-bit of the intermediate value of $g(x)$ must be zero. To find one solution of $g(x)$, it is necessary to compute $2^r$ times in this case. The time complexity required by our PoW is $2^r$ times. Figure 8 shows the change in time complexity with the limit of $r$-bit in the case of $n = 256$. For example, when $n = 256$ and $k_1 = k_2 = 128$, the time complexity is $2^{128}$. If $r = 64$, the time complexity is $2^{192}$, i.e., that is $2^{64}$ times larger.

6.3 Tuning Time and Memory Complexity

First, we adjust the value of $k_2$ to specify memory complexity. We then specify time complexity by limiting the intermediate value to the value of $r$. This process makes it possible to uniquely determine the time and memory complexity required by the PoW. It should be careful not to break the conditions in Eq. (5)-(7) not to allow time complexity to exceed $2^n$. If the parameters do not satisfy them, then the attacker can use brute force attacks and F-MITM.

6.4 Practical Parameters

We give an example parameter such that our scheme is equivalent in time complexity to Hashcash, and consumes memory in a realistic range.

Hashcash uses SHA-2 or RIPEMD, which is a hash function with an output length of 256 bit[5]. In the case of Hashcash, the miner has to search for a nonce where the hash value is less than the target value. The target is adjusted every 2016 blocks generated so that it takes about 600 seconds to generate one block depending on the hash
Table 1 Security requirements satisfied by our and other schemes

|                           | Hashcash [5] | Equihash [6] | MTP [1] | Our Scheme |
|---------------------------|--------------|--------------|---------|------------|
| Amortization-free         | Yes          | Yes          | Yes     | Yes        |
| Asymmetry                 | Yes          | Yes          | Yes     | Yes        |
| Flexibility               | No           | Insufficient | Yes     | Yes        |
| Progress-free             | Yes          | Yes          | Yes     | Yes        |
| Steep time-memory tradeoff| No           | Yes          | Yes     | Yes        |
| Parallelism               | No           | Constr.      | Constr. | Constr.    |
| Optimization-free         | Yes          | Yes          | Insufficient | Yes      |

Table 2 Practical parameters for our scheme

| n  | k1 | k2 | time | memory |
|----|----|----|------|--------|
| 91 | 60 | 60 | 31   | $2^{29}$ ($\approx 2.5$GB) |
| 98 | 65 | 65 | 33   | $2^{55}$ ($\approx 98$GB)  |
| 106| 70 | 70 | 36   | $2^{70}$ ($\approx 848$GB) |
| 113| 75 | 75 | 38   | $2^{75}$ ($\approx 3.5$TB) |

rate of the whole network [13], [14]. At June 18, 2021, it requires about $2^{76.1}$ hashing to generate one block [15]. Table 2 shows each parameter and its time and memory complexity.

For the parameters in Table 2, we should use AES, which has both block length and key length of 128 bit. When the key length is $k_1 = 40$ and $k_2 = 32$, the most significant 88 bits and 96 bits of each key must be set to 0.

7. Discussion on Other Properties

In this section, we consider whether our scheme satisfies other properties, and then compares our schemes with existing schemes of Hashcash [5], Equihash [6], and Merkle Tree Proof [1].

7.1 Amortization-Free

To satisfy the property of amortization-free, the computational cost to find $q$ solutions should be $q$ times higher than that of finding one solution. Since our scheme gives the plaintext and ciphertext randomly each time, the results of previous calculations cannot be used. Therefore, our scheme meets amortization-free.

7.2 Asymmetry

To satisfy the property of asymmetry, the proofs require a certain amount of time and memory usage, while the verification needs only a few resources. Our scheme is flexible to determine time and memory complexity. However, the verifier only encrypts with the keys sent by the prover and check whether the output matches the ciphertext. Thus, our scheme fully satisfies asymmetry.

7.3 Steep Time-Memory Tradeoff

To satisfy the property of steep time-memory tradeoff, the amount of computational complexity should exponentially increase when the amount of memory is reduced. This property is important for ASIC resistance. Figure 6 shows that our scheme satisfies this requirement.

7.4 Flexibility

To satisfy the property of flexibility, time and memory must be independently adjustable. As mentioned in Sect. 5.2.3, we can flexibly adjust the time and memory complexity. In addition, the time complexity is dependent on $r$ and memory complexity is dependent on $k_2$. Therefore, we can adjust the time and memory complexity separately. Our scheme perfectly satisfies the property of flexibility.

7.5 Optimization-Free

To satisfy the property of optimization-free, the prover must use the optimized algorithm. Generally speaking, it is difficult for schemes based on symmetric-key cryptography to judge whether they satisfy optimization-free because the security of symmetric-key cryptography is not reduced on mathematical problems, unlike public-key cryptography. Indeed, some attacks on the first version of MTP have been proposed [7]. On the other hand, Rijndael won the contest to determine AES and is not critically vulnerable so it has been used since 2000. Thus, we consider that our scheme based on AES sufficiently satisfies the optimization-free.

7.6 Progress-Free

To satisfy the property of progress-free, the number of solutions found in a given time must approximate the Poisson process. Since AES can be regarded as a random oracle, it can also approximate a Poisson process. Therefore, we expect that our scheme achieves the property of progress-free.

7.7 Parallel-Unfriendly

As discussed in [1], [6], parallelism is restricted in our scheme due to the memory bandwidth growth in parallel implementations. Therefore, we expect that our scheme achieves the property of parallel-unfriendly.

7.8 Comparison with Existing Schemes

Table 1 shows the comparison between our scheme, Hashcash [5], Equihash [6], and MTP [1]. Hashcash does not satisfy the properties of flexibility, steep time-memory tradeoff, and parallel-unfriendly. Equihash [6] does not sufficiently
achieve flexibility. Since several variants of the first version of MTP [1] were broken [7], MTP [1] does not satisfy the optimization-free. Our scheme satisfies all of the required properties for PoW.

8. Conclusion

In this paper, we proposed the new PoW scheme based on the key recovery problem of 2CBC. By choosing the appropriate parameters such as block size and key sizes of underlying block ciphers, we could fully control the required time complexity and memory complexity for efficiently solving the problem. In particular, the required memory and time complexity were tunable independently according to requirements that depend on target applications and the progress of computational power. As a result, our scheme could force the use of memory to achieve ASIC resistance.

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] A. Biryukov and D. Khovratovich, “Egalitarian computing,” T. Holz and S. Savage, editors, 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, Aug. 10-12, 2016, pp.315–326, USENIX Association, 2016.

[2] B. Chen and S. Tessaro, “Memory-hard functions from cryptographic primitives,” A. Boldyreva and D. Micciancio, editors, Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, Aug. 18-22, 2019, Proceedings, Part II, Lecture Notes in Computer Science, vol.11693, pp.543–572, Springer, 2019.

[3] D. Boneh, H. Corrigan-Gibbs, and S.E. Schechter, “Balloon hashing: A memory-hard function providing provable protection against sequential attacks,” J.H. Cheon and T. Takagi, editors, Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, Dec. 4-8, 2016, Proceedings, Part I, Lecture Notes in Computer Science, vol.10031, pp.220–248, 2016.

[4] A. Biryukov, D. Dinu, and D. Khovratovich, “Argon2: New generation of memory-hard functions for password hashing and other applications,” IEEE European Symposium on Security and Privacy, EuroS&P 2016, Saarbrücken, Germany, March 21-24, 2016, pp.292–302, IEEE, 2016.

[5] A. Back, “Hashcash—a denial of service counter-measure,” 2002.

[6] A. Biryukov and D. Khovratovich, “Equihash: Asymmetric proof-of-work based on the generalized birthday problem,” Ledger, vol.2, pp.1–30, 2017.

[7] I. Dinur and N. Nadler, “Time-memory tradeoff attacks on the MTP proof-of-work scheme,” J. Katz and H. Shacham, editors, Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, Aug. 20-24, 2017, Proceedings, Part II, Lecture Notes in Computer Science, vol.10402, pp.375–403, Springer, 2017.

[8] U.M. Maurer and J.L. Massey, “Cascade ciphers: The importance of being first,” J. Cryptol., vol.6, no.1, pp.55–61, 1993.

[9] R.W. Floyd, “Nondeterministic algorithms,” J. ACM, vol.14, no.4, pp.636–644, 1967.

[10] I. Nikolic and Y. Sasaki, “A new algorithm for the unbalanced meet-in-the-middle problem,” J.H. Cheon and T. Takagi, editors, Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, Dec. 4-8, 2016, Proceedings, Part I, Lecture Notes in Computer Science, vol.10031, pp.627–647, 2016.

[11] M. Hellman, “A cryptanalytic time-memory tradeoff,” IEEE Trans. Inf. Theory, vol.26, no.4, pp.401–406, 1980.

[12] N. Mouha and A. Luykx, “Multi-key security: The even-mansour construction revisited,” Annual Cryptology Conference, vol.9215, pp.209–223, Springer, 2015.

[13] K.J. O’Dwyer and D. Malone, “Bitcoin mining and its energy footprint,” 25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communities Technologies (ISSC 2014/CIICT 2014), pp.280–285, 2014.

[14] D. Fullmer and A.S. Morse, “Analysis of difficulty control in bitcoin and proof-of-work blockchains,” 2018 IEEE Conference on Decision and Control (CDC), pp.5988–5992, IEEE, 2018.

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