How to generate transparent random numbers using blockchain

Yuto Ehara[1]
Graduate School of Science
Chiba University
Yayoicho 1-33, Inage, Chiba 263-8522, Japan.

Mitsuru Tada
Institute of Media and Information Technologies
Chiba University
Yayoicho 1-33, Inage, Chiba 263-8522, Japan.
E-mail: m.tada@faculty.chiba-u.jp

***Abstract*—Nowadays, we can enjoy many internet applications including online lotteries and games. The results of lotteries, card games, dice throwing and so on, depend upon the values of random numbers generated under some rules. However, we cannot verify the correctness for the given random numbers, that means, we have no means to convince ourselves that the given random numbers have indeed been correctly generated, using a pre-determined algorithm, and without being intentionally perverted by anyone. We say that such random number generation can provide 'transparency'. It is quite important that random numbers given online can convince us that those are correctly generated. In other words, the transparency of online random number generation is quite important. Sako et. al. [5] have given a random number generation scheme applicable to the game 'Backgammon', in which transparency of blockchain enables us to verify the correctness for used random numbers after the game ended. In that scheme, however, the server has to be a trusted party, and it means dishonest generation of random numbers is possible by making one player and the server corrupted. In this paper, we present a transparent random number generation scheme. To be sure that a few trusted parties appears in the description of the scheme, but by implementing such trusted ones with public blockchain actually operated, our scheme can generate transparent random numbers without requiring any trusted parties.**

I. Introduction

Nowadays, random numbers are used in many computer applications. To be sure that random numbers are quite important for the security of encryption and digital signature schemes, but here we would like to focus on random numbers used for entertainments such as lotteries and games in a network.

Imagine that we enjoy a lottery in a network application, and that we can get various gifts corresponding to the result of the lottery, which is determined according to some random number(s). If we get a *good* gift or one we *aim*, then we may be satisfied. But otherwise, we may suspect if the system manager has shown some dishonest result so that we feel we want to try again.

Again, imagine that we play a match-type card game via network. During the game, we may often pull some cards. If those pulled card are all acceptable for our hand, and if we win, then we may be satisfied. But otherwise, if our pulled cards are entirely unfavorable, and if we lose thereby, we may suspect if the system manager or our competitor may control the cards so that we do not win.

[1]The current affiliation of the first author is: Media Entertainment Japan Inc., THE WORKERS & CO 2F, Higashiazabu 1-4-2, Minato-ku, Tokyo 106-0044, Japan. E-mail: y.ehara@mejy.jp

The problem is that we cannot verify that the result is indeed derived by a *correct* manner. If we can convince ourselves that the result is derived from some seed(s) which is not under anyone's control, then we never suspect anyone, and we only lament our misfortunes. However, even if the system manager does execute the correct algorithm using a seed under no one's control, if we cannot verify it, then such random number generation is not said to provide *transparency*.

In this paper, we say *transparent* random numbers to mean ones under the conditions:

(T1) The value of the random number is derived using some pre-determined (public) function F;
(T2) No one can control the value of the random number by regulating the arguments (i.e. the seeds) of the function F;
(T3) Everyone can verify (T1) after the value of the random number is shown.

From the condition (T2), the function F must be, at least, one-way and must provide the property like collision-resistance. Here, we suppose that two entities $\mathcal{A}, \mathcal{B}$ determine the seeds a, b, respectively, and that the random number α is defined as $\alpha \stackrel{\text{def}}{=} F(a, b)$. If some trusted entity $\mathcal{T}$ is provided, then we have only to make $\mathcal{T}$ receive a, b from $\mathcal{A}, \mathcal{B}$ keeping a secret against $\mathcal{B}$ and b against $\mathcal{A}$, and have only to make $\mathcal{T}$ compute α as above and show (α, a, b) to $\mathcal{A}, \mathcal{B}$. Furthermore, make $\mathcal{T}$ receive the commitments on a, b before showing α, and $\mathcal{A}$ or $\mathcal{B}$ cannot object on the result i.e. the value of α shown by $\mathcal{T}$. Based on such an idea, the random number generation scheme [5] has been constructed. In [5], blockchain is used as a safe-deposit for the commitments and other seeds. Since $\mathcal{T}$ must be a trustee[2], $\mathcal{A}$ or $\mathcal{B}$ cannot control the value of a (or b) so that the result α is acceptable for $\mathcal{A}$ (or $\mathcal{B}$). However, that means that if $\mathcal{A}$ can collude with the server, then the random number generation may run as $\mathcal{A}$ desires.

In this paper, we present a random number generation scheme using blockchain so that we can get transparent random numbers without any trusted entities. In the proposed scheme we describe in Section IV, two entities appear being trusted, but transparent random number generation without

[2]In [5], $\mathcal{T}$ here is named 'the server'.

any trustees can be constructed by implementing the scheme using a public blockchain which adopts a 'Proof of Work'-type consensus algorithm, and which enables a smart contract, like Ethereum[3].

This paper is organized as follows: In Section II, we review the concept of blockchain, some terminologies on blockchain and an example of blockchain which enables a smart contract. In Section III, we review the random number generation scheme by [5]. In Section IV, we show our proposed random number generation scheme using blockchain. In Section V, we give some discussion on the proposed scheme. Finally, Section VI concludes this paper.

II. Blockchain

Before describing the existing scheme[5] and our proposed scheme, we review the concept of blockchain and some associating terms on blockchain. After then, we give a brief description on Ethereum[3].

A. The structure of blockchain

Blockchain is a technique to make a distributed ledger by sharing data among computers interspersed over the world, and it can realize cryptocurrencies such as Bitcoin[4] and Ethereum[3]. It has the following advantages[1]:

- It is quite hard to ruin or tamper with the stored data;
- It can provide a zero-downtime;
- It can provide transparency on the stored data;
- It can be constructed at relatively low cost.

A blockchain in which any entities can participate like one for Bitcoin, is called a *public* blockchain, and on the other hand, a blockchain in which only permitted entities can participate is called a *consortium* or *private* blockchain[3].

In general, a block in a blockchain includes the following components:

(C1) A time-stamp proving the time when the block is created;
(C2) The hashed value of the previous block;
(C3) The value of the nonce (See Section II-B);
(C4) Transaction data;
(C5) Other (auxiliary) data necessary for creating the block.

A block is connected with the previous one, and will be connected with one created next. Blocks look like a chain as seen in Fig. 1.

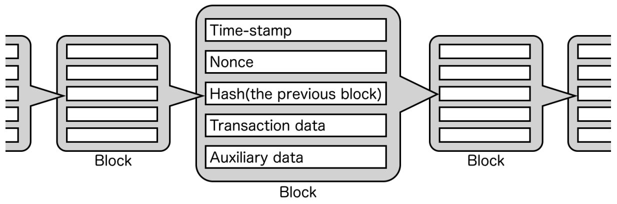

Fig. 1. Blockchain

[3]In general, a consortium blockchain is managed by plural organizations, and a private blockchain is managed by a single one.

A sequence of blocks yields a hash-chain since each block includes the hashed value of the previous one. Therefore, an adversary who attempts to tamper with the transaction data included some past block, has to re-create all the descendant blocks associating with the tampered one.

B. Consensus algorithm

A *miner* is an entity who has a right to create the next block, and is chosen with a pre-determined *consensus algorithm*. In general, a consensus algorithm is for mutual recognition among the participants on a certain result in a network with a time-lag like P2P network. *Proof of Work* (PoW, for short) is one of the most popular consensus algorithms, and is used for Bitcoin[4] and Ethereum[3]. PoW is a mechanism for verifying no dishonesty by compelling miner candidates[4] to execute some pre-determined operation, and the operation has the following properties:

- The operation requires much computing effort for the result;
- But the verification of the result above requires little computation power.

Such an operation can be realized with the randomness of an appropriate hash function like $\mathsf{SHA256}$, as follows: At the timing the transaction data is broadcast, each miner candidate tries a random value as the nonce (the block component C3) until the hashed value x of the block with the nonce value satisfies some pre-determined condition such as $x < t$, where t is a pre-determined target value and denoted by *Target* hereafter. That means each miner candidate has to find a (C3) component n which satisfies

$$\mathsf{hash}(\text{the block including } n) < \textit{Target}, \tag{E1}$$

just after all the components except for (C3) have been fixed.

Let us see the case of Bitcoin. Since $\mathsf{SHA256}$ is adopted as $\mathsf{hash}(*)$, the hashed value of a block is in $[0, 2^{256} - 1]$, and it is hard to invert some hashed value less than *Target* in order to get some valid n. Hence each miner candidate has to find an n satisfying (E1) with repeated trial for n. Since the probability of finding a valid n with only one trial is $\textit{Target}/2^{256}$, and since the value of *Target* for Bitcoin on March 31st in 2018 is about $2^{182.33}$, the probability above is about $1/2^{74}$ which is quite small. The miner candidate who find a valid nonce can be the miner of the very block.

C. Ethereum

Ethereum[3] is a decentralized platform that runs smart contracts, released in 2015, in which a blockchain with PoW is adopted, and in which the cryptocurrency 'ether' is realized. Furthermore, unlike Bitcoin, it enables not only exchange of cryptocurrency but also execution of a smart contract defined by a user.

[4]Any entities can be miner candidates and try to be the miner.

III. The previous scheme

In this section, we review the scheme presented by [5], which enables us to verify, using blockchain, the correctness for the random numbers generated in some online game.

A. The abstract of [5]

In [5], we can see how to generate transparent random numbers at some online game[5], using blockchain. The scheme given by [5] can convince the game players that the random numbers they have used at the game are not unfairly controlled. Still more, [5] shows that the implementation of the scheme applied to Backgammon works with no problem, using the blockchain for Bitcoin. However, as we see later, corrupting the server given below enables dishonest random number generation.

B. The scheme [5]

Here we see the process[6] of the random number generation scheme[5]. The participants are two game players ($\mathcal{P}_1$ and $\mathcal{P}_2$) and the server $\mathcal{S}$. Let $\mathsf{hash}(*)$ be an appropriate hash function. Hereafter we call, a *random number*, a consequent value which is generated along the random number generation scheme, and call, a *seed*, a value which a participant generates independently to other participants in order to compute a random number along the scheme.

(1) The server $\mathcal{S}$ generates seeds s_1 and s_2, and registers those hashed values $\mathsf{hash}(s_1)$ and $\mathsf{hash}(s_2)$ to the blockchain as the commitments.
(2) Each player P_i $(i = 1, 2)$ determines a seed r_i, and sends its hashed value (i.e. $\mathsf{hash}(r_i)$) to $\mathcal{S}$.
(3) $\mathcal{S}$ forwards each $\mathsf{hash}(r_i)$ to the other player (i.e. $\mathcal{P}_{3-i}$), registering both $\mathsf{hash}(r_i)$ to the blockchain.
(4) Each $\mathcal{P}_i$ receives the hashed value by the other player, and sends r_i to $\mathcal{S}$.
(5) $\mathcal{S}$ makes the game start after verifying the consistencies for both salts r_i and the hashed value $\mathsf{hash}(r_i)$.
(6) When a dice throw is first required, the server $\mathcal{S}$ shows $x(1)$ computed as $x(1) \overset{\text{def}}{=} \mathsf{hash}(r_1, r_2, y(1))$ where $y(0) \overset{\text{def}}{=} \mathsf{hash}(r_1, r_2, s_1, s_2)$ and $y(1) \overset{\text{def}}{=} \mathsf{hash}(y(0))$.
If the random number generation (the dice throw) executed $(N-1)$ times, $\mathcal{S}$ has shown $x(1), x(2), \dots, x(N-1)$ and keeps $y(0), y(1), \dots, y(N-1)$. When the N-th dice throw, $\mathcal{S}$ computes $y(N) \overset{\text{def}}{=} \mathsf{hash}(y(N-1))$ and $x(N) \overset{\text{def}}{=} \mathsf{hash}(r_1, r_2, y(N))$, and outputs $x(N)$ as the required random number.
(7) When the game is ended, $\mathcal{S}$ sends s_1, s_2, r_1 and r_2 to both players.
(8) Each player verify that the hashed values of s_1, s_2, r_1 and r_2 coincide with those registered to the blockchain, and that each random number (dice throw) has indeed been figured out using s_1, s_2, r_1 and r_2.

[5]In [5], 'Backgammon' is adopted as the online game.
[6]Note that we use different notations from those [5] uses.

C. The problem on [5]

The server $\mathcal{S}$ obtains the seeds (r_1 and r_2) of the players at Step (4), and hence $\mathcal{S}$ can, in advance, see the whole of the random numbers (the dice throws) that will be executed at Step (6). That means that if the player $\mathcal{P}_1$ colludes with the server, then $\mathcal{P}_1$ can obtain any future dice throws from the server, and then $\mathcal{P}_1$ can go on with the game favorably with $\mathcal{P}_2$. Therefore the server in [5] should be a trusted third party.

IV. Proposed scheme

As seen in the previous section, in the scheme [5], the server can be verified generating random numbers correctly, but must be a trusted third party not colluding to any players. The authors prefer that transparent random number generation is possible even if there is no trusted party, and present a scheme which enables such random number generation.

A. Participants

In the proposed scheme, the participants are following:

- The users[7]: $\mathcal{U}_1, \mathcal{U}_2, \dots$;
- The service systems[8]: $\mathcal{S}_1, \mathcal{S}_2, \dots$;
- The random number generator: $\mathcal{G}$;
- The blockchain holder: $\mathcal{E}$.

For the sake of simplicity, we set both numbers of users and service systems to be one, and hence, we write as $\mathcal{U}$ to denote the user, and $\mathcal{S}$ to denote the service system.

The user $\mathcal{U}$ has a unique address a, and the service system $\mathcal{S}$ also has a unique address b, such that those are distinguished in the network. Then we suppose that $\mathcal{U}$ knows the address b of $\mathcal{S}$.

The random number generator $\mathcal{G}$ executes the algorithm which is open (that means, known to every entity) and registered to the blockchain, and is implemented as a smart contract on a blockchain. $\mathcal{G}$ can execute the following two kinds of queries to the blockchain holder $\mathcal{E}$ given below.

- $\mathsf{Reg}(\mathit{info})$: Registration of the received data info to the blockchain $\mathcal{E}$ holds;
- $\mathsf{AskNonce}()$: Inquiry, to $\mathcal{E}$, for the nonce at the blockchain to come next.

The blockchain holder $\mathcal{E}$ provides a blockchain to which at least every user and every service system can refer, receives requests of $\mathsf{Reg}(*)$ and $\mathsf{AskNonce}()$ from $\mathcal{G}$, and correctly executes those requests.

In the description given below, $\mathcal{G}$ and $\mathcal{E}$ are modeled to be trusted, that means, those never collude with any other parties or never break what the smart contract says. However, by implementing those with a public blockchain which enables smart contracts like Ethereum[3] which adopts a consensus algorithm like PoW requiring huge computational power for

[7]Since the proposed scheme does not assume any situation such as games and lotteries, we say 'a user' to mean an entity that may enjoy using the results of online random number generation.
[8]A service system is an entity that may offer giveaways to users according to the values of random numbers generated (i.e. online lottery), or that may manage players in online games using the values of random numbers generated.

the mining, no trusted party for transparent random number generation turns out be required.

B. The requirements for transparent random number generation for network applications

The authors think that the following conditions are required for transparent random number generation for online applications (especially, for entertainments such as games and lotteries).

(R1) The value of the random numbers generated should depend upon both the seed by the user and that by the service system.
(R2) Neither the user nor the service system can guess or control the value of the random numbers generated, with a non-negligible probability.
(R3) The user and the service system can verify the correctness of the random numbers, i.e., that the random numbers generated have been computed using the correct parameters and the correct algorithms.

The requirement (R2) corresponds to the condition (T2) in Section I, and the requirement (R3) corresponds to the conditions (T1) and (T3). The requirement (R1) is possibly dispensable for the security and the transparency, but the authors think (R1) is necessary for feeling of satisfaction of a user, especially a user enjoying entertainments.

C. Basic idea

In [5], the arguments of the used hash function are, seeds participants(i.e. players) choose and that the server chooses. Therefore, participants have to send, besides the seeds they choose, their commitments, before the random numbers are generated.

In our scheme, the arguments includes the nonce which is found for a new block creation for a blockchain with a PoW-type consensus algorithm. Such a nonce cannot be controlled by any entities since it is found with difficulty by many computers that are rivals of each other. Hence, in case that only one user and only one service system participate the random number generation, no other parameter is required, that means, a nonce (or its hashed value) will do.

However, since it takes some time for finding a new nonce, if the user requires two random numbers together, two numbers of the identical value are returned. Hence, our scheme adds a serial number to the arguments of the hash function.

However, if that is done so even if plural (for example, two) users participate, since they do not know which random numbers are for them, our scheme registers the relationship between the serial number and the addresses of the participants, and adds the addresses to the arguments of the hash function. Furthermore, for (R1), we add seeds chosen by users and service systems, besides the parameters given above, that is, the nonce, the serial number and the addresses of the user and the service system.

D. The processes of the proposed scheme

The proposed random number generation scheme consists of three procedures as follows:

(P1) The user registration procedure;
(P2) The service system registraion procedure;
(P3) The lottery procedure.

Any procedure can be executed repeatedly, under the condition that both the user registration procedure and the service system registraion procedure must be executed at least once before the lottery procedure is executed. The three procedures are following.

The user registration procedure

The user $\mathcal{U}$ registers his address a and a seed used for computing random numbers, to the blockchain.

(u1) The user $\mathcal{U}$ determines a seed r, and sends, to the random number generator $\mathcal{G}$, his address a and the seed r.
(u2) $\mathcal{G}$ receives a and r from $\mathcal{U}$, and registers those executing $\mathsf{Reg}(a, r)$.

The service system registraion procedure

The service system registers its address b and a seed used for computing random numbers, to the blockchain.

(s1) The service system $\mathcal{S}$ determines a seed p, and sends, to the random number generator $\mathcal{G}$, its address b and the seed p.
(s2) $\mathcal{G}$ receives b and p from $\mathcal{S}$, and registers those executing $\mathsf{Reg}(b, p)$.

The lottery procedure

The user $\mathcal{U}$ requests a random number generation, and $\mathcal{U}$ and the corresponding service system get the value of the random number generated.

(d1) The user $\mathcal{U}$ sends a request to the random number generator $\mathcal{G}$ by sending his address a and the address b of the service system $\mathcal{U}$ applies to.
(d2) $\mathcal{G}$ makes a (unique) identification number sn (which may be a serial number).
(d3) $\mathcal{G}$ finds the latest r in the blockchain which is associated with the address a, and finds the latest p in the blockchain which is associated with the address b. Then $\mathcal{G}$ registers sn, a and b executing $\mathsf{Reg}(\mathit{sn}, a, b)$.
(d4) $\mathcal{G}$ executes $\mathsf{AskNonce}()$ to get the nonce n from $\mathcal{E}$ at the timing when the next block is generated.
(d5) $\mathcal{G}$ computes $\alpha \stackrel{\text{def}}{=} \mathsf{hash}(r, p, \mathit{sn}, a, b, n)$[9].
(d6) $\mathcal{G}$ sends, to $\mathcal{U}$ and $\mathcal{S}$, sn and α. Here, α is the random number generated for the identification number sn.
(d7) $\mathcal{U}$ and $\mathcal{S}$ can verify whether the result α is indeed computed using r, p, sn, a, b and n.

[9]Here we suppose $\mathsf{hash}(*)$ to be a world-widely used hash function such as $\mathsf{SHA256}$, but $\mathsf{hash}(*)$ may be a $\mathsf{SHA256}$-hashing after some operations such as bit-permutations and other hashings, instead of single $\mathsf{SHA256}$-hashing. Then we can write as $\alpha \stackrel{\text{def}}{=} \mathsf{SHA256}(f(r, p, \mathit{sn}, a, b, n))$ for some operation f. However, since f is not essential for randomness of α (and for security of α), we write $\alpha \stackrel{\text{def}}{=} \mathsf{hash}(\cdots)$ to avoid complicatedness.

E. Are the requirements satisfied?

We can say that the proposed random number generation scheme given in the previous subsection indeed satisfies all the requirements (R1), (R2) and (R3) we have seen in Section IV-B, from the following facts:

(1) The random number α output finally, depends upon the seeds r and p which are chosen by the user $\mathcal{U}$ and the service system $\mathcal{S}$, respectively. This implies (R1).
(2) The value of the random number α depends upon the value of the nonce n, and it is generally computationally hard to guess the value of the nonce n. If the hash function $\mathsf{hash}(*)$ can play a role of a *secure* pseudorandom function, then the probability that an arbitrary entity who cannot, with a non-negligible advantage, guess the value of the next nonce n, is negligibly small in general. Thus (R2) follows.
(3) The step (d7) indicates (R3).

V. Discussion

Here, on the proposed scheme, we see the performance, and discuss on the efficiency and the security.

A. Security

Here we consider the security of the proposed random number generation scheme. Since the proposed scheme aims to enable no one to control the random numbers generated, it is appropriate to suppose that an adversary may intend to guess the value of the random number generated next. The random number α generated is defined as $\alpha \overset{\text{def}}{=} \mathsf{hash}(r, p, sn, a, b, n)$, described in Section IV-D. If we adopt the hash function $\mathsf{hash}(*)$ being secure enough, i.e. if the function $\mathsf{hash}(*)$ has the properties such as one-wayness and collision resistance, then guessing the value of the random numbers generated is as hard as guessing the value of the nonce. The value n of nonce is something which satisfies

$$\mathsf{hash}(T, n) < \mathit{Target},$$

for some pre-determined value *Target* and the transaction data T[10].

For example, on March 31st in 2018, the difficulty[2] for Bitcoin is

$$3,494,288,842,679 \doteqdot 2^{41.67},$$

which derives $\mathit{Target} \doteqdot 2^{182.33}$. This means that miners are required to find some value n with $\mathsf{hash}(T, n) < 2^{182.33}$, i.e. a value n which derives a bit-string that has a leading $0 \cdots 0$ (74 zeros, since a 256-bit hash function is used for Bitcoin). If the 256-bit hash function ($\mathsf{SHA256}$ in this case) is secure enough, then the probability that we can obtain a valid nonce with only one trial is almost $1/2^{74}$. Many computers all over the world, with all their strength, can find such a value about in ten minutes. An adversary with reasonable computation power cannot do such a thing. Actually, the expected value E of the time required for finding a valid nonce is

$$E = xt + (1-x)x \cdot 2t + (1-x)^2 x \cdot 3t + \cdots = \frac{x}{t},$$

where x is the probability of finding a valid nonce with one hashing trial, and where t is the required time for one hashing. In case with $x = 1/2^{74}$ and $t = 3.3 \times 10^{-6}$ (sec)[11], the expected value is about 6.29×10^{16} (sec), which is almost 2000 million years.

Thus, by using a blockchain which are known to adequate many persons, and for which adequate many computers participate for the mining, the security of the proposed random number generation scheme can be actually guaranteed.

B. Efficiency

In general, it costs some money (such as bitcoin or ether) to register some information to a blockchain. Roughly speaking, such a cost depends upon the times of the registration to the blockchain in case that the data size registered to the blockchain is almost constant.

Let us suppose that in the proposed scheme, the user registration procedure and the service system registraion procedure are executed only once each, and that the lottery procedure is executed N times (i.e. that N random numbers are generated). Then communication occurs $(4 + 6N)$ times, and registration to the blockchain occurs $(2 + N)$ times.

VI. Conclusion and future work

Transparent random number generation is important for network entertainments such as games and lotteries not so that users have doubts on the results of those entertainments. In this paper, we have presented a transparent random number generation scheme using blockchain so that we can get transparent random numbers without any trusted entities.

However, we have a few problems on the performance and the scalability of our scheme. Actually, since it costs several minutes to get a new nonce when we use a public blockchain with PoW, then we have to spend the same amount of time to get a new random number. That means our scheme may be applicable to lotteries, but it may be hard to apply to online games such as interactive match-type card games. We would like to make the improvement of the performance our future work.

[10]Actually, for example, in case Bitcoin, a nonce is determined after the argument (T, n) is hashed twice, is permutated under some rule, and is hashed again[6]. That means the nonce is something satisfies $\mathsf{hash}(f(T, n))$, where f is some pre-determined function.

[11]This t is an average time with a 2.7GHz-CPU Core-i7 Linux machine that one of the authors possesses.

References

[1] A. M. Antonopoulos: Mastering Bitcoin, Oreilly & Associates Inc., 2017.
[2] Blockchain: Difficulty.
`https://blockchain.info/ja/charts/difficulty`
[3] Ethereum Foundation: "Ethereum project".
`https://www.ethereum.org/`
[4] Satoshi Nakamoto: "*Bitcoin: A peer-to-peer electronic cash system*", 2008.
`https://bitcoin.org/bitcoin.pdf`
[5] K. Sako and K. Iguchi: "*Fair coin flipping for online games using blockchain*", Proceedings of The 2017 Symposium on Cryptography and Information Security (SCIS2017), 1F2-4, (in Japanese), 2017.
[6] S. Yamazaki, S. Azuchi and S. Tanaka: Blockchain programming - Introduction to cryptocurrency, Kodansha, (in Japanese), 2017.