Blockchain-Based Ubiquitous Code Ownership Management System without Hierarchical Structure

Hirotsugu Seike [1], Takeo Hamada [1], Takahiro Sumitomo [1], Noboru Koshizuka [1, 2]
[1] Interfaculty Initiative in Information Studies, The University of Tokyo, Tokyo, Japan
[2] YRP Ubiquitous Networking Laboratory, Tokyo, Japan
Email: {hirotsugu.seike, takeo.hamada, takahiro.sumitomo, noboru}@koshizuka-lab.org

***Abstract*—The ubiquitous ID (uID) architecture provides dynamic and flexible context-awareness frameworks necessary in ubiquitous computing. Its basic idea is to assign 128 bit unique identifiers, each of which is called a ucode (ubiquitous code), to real-world entities. To guarantee the uniqueness of ucodes, current ucode ownreships are maintained by a hierarchical structure like DNS. In this traditional system, processing ucode transactions between organizations is cumbersome and inefficient because the reliability of owner information is dependent on individual organizations.**

To solve this problem, we designed a ucode ownership management system on a blockchain which is a decentralized platform. In this system, we proposed a blockchain-based ucode allocation method, which requires a simple procedure and only one transaction. This proposed method is user-friendly and efficient compared with ucode allocation methods to which we simply apply current blockchain-based name registration methods requiring a complex procedure and at least two transactions.

Finally, we conducted a case study about these ucode allocation methods built on the Ethereum blockchain. From this result, we compared the usability and security of each ucode allocation method in Ethereum.

***Index Terms*—ubiquitous code, ownership management, unique identifier allocation, blockchain, Ethereum**

I. Introduction

To recognize real-world contexts is important in ubiquitous computing that supports our daily lives. For this reason, the ubiquitous ID (uID) architecture [1] was proposed to provide dynamic and flexible context-awareness frameworks. Its basic idea is to assign 128 bit unique identifiers to various objects, spaces and concepts in the real world because distinguishing those is essential to realize context-awareness. Each of these identifiers is called a ucode (ubiquitous code) [2] and has been standardized by ITU-T [3]. To guarantee that no two targets have the same ucode, current ucode ownerships are maintained by a hierarchical structure like DNS. In this traditional system, since the reliability of owner information is dependent on each management organization, the process of ucode transactions between organizations are inefficient and cumbersome. To handle such a problem which a hierarchical structure has, blockchain-based naming systems [4, 5, 6] that can manage domain name ownerships were proposed and are widely used. However, there are no blockchain-based systems that can manage ucode ownerships.

Blockchain technology was first introduced by S. Nakamoto for Bitcoin [7], which is a digital currency, and currently is applied to various fields beyond exchanging money. Blockchains can provide cryptographically secure, consistent, decentralized and append-only ledgers on respective P2P networks without central authorities. This enables us to re-build applications that could be previously realized only through trusted third-parties [8]. On the other hand, applications on a blockchain suffer from two scalability problems: low transaction throughput and high latency [9]. There is a trade-off between performance and centralization. Developers must consider this problem especially using fully public blockchains, for example Namecoin [4], Bitcoin [7] and Ethereum [10].

In order to manage ownership of unique identifiers, such as domain names and ucodes, it is necessary to prevent squatting and front-running, both of which are mentioned in [11]. In blockchains, these two issues can easily occur due to no centralized control and high transaction latency. To mitigate them, Namecoin [4] and Blockstack [5], each of which provides a blockchain-based domain name service, adopts the mechanisms of destroying coins and registering a name by a two-phase commit process. However, this name registration method is not user-friendly [12] and requires two transactions. Ethereum Name Service (ENS) [6], which is a distributed naming system built on top of Ethereum, uses the auction-based name registration method defined as EIP 162 [13]. In this auction-based name registration method, since a user who bids the highest amount can get the target name, squatting and front-running do not occur in principle. However, it works complicatedly and needs 3 or more transactions.

In this paper, we design a ucode ownership management system on a blockchain. In this system, we propose a blockchain-based ucode allocation method which requires a simple procedure and only one transaction. This method is realized by automatically generating a unique number and assigning it to a particular ucode's field at every time ucode allocation. This method is user-friendly and efficient compared with ucode allocation methods to which current blockchain-based name registration methods are simply applied. These conventional methods require a complex procedure and at least two transactions. We conduct a case study about our proposal built on Ethereum. To calculate the mean transaction confirmation time that affects the characteristics of blockchain-based systems, we run a simulation by a queuing model which takes into account mining process by using Ethereum block and transaction data. From this result, we compare the usability and security of each ucode allocation method.

II. Related Work

In this section, we first explain an overview of blockchains and general requirements to build blockchain-based ownership management system for unique identifiers, such as domain names or ucodes. Next, we mention three blockchain-based naming systems and point out defects of current name registration methods.

A. Overview of blockchains and general requirements to build resource ownership management system

Blockchains can provide decentralized and append-only ledgers that are writable publicly. The ledger on a blockchain can be updated only by transactions which are atomic, consistent, isolated and durable (ACID) [14]. These transactions are organized as blocks. The blockchain database is updated on each block by executing its multiple transactions in a algorithmically determined order. Therefore, sharing a series of blocks, which is also known as a blockchain, enables us to have a distributed database. To realize that network nodes have the same sequential blocks, many consensus algorithms, such as PoW [7], were proposed. See [15] for further details on how to reach consensus by these algorithms.

Blockchain-based naming systems can manage ownership of unique identifiers, called domain names. To determine name owners, each domain name is associated with that owner's public key, like PGP [16]. These naming systems have functions to mitigate two problems: squatting and front-running [11]. Squatting is the action of occupying unique identifiers, such as domain names and ucodes, without intention to use them. Front-running, which is analogous to domain name front-running [17], is the action to register a unique identifier which is about to be newly registered by another user through intercepting his registration queries. In blockchains, front-running is caused by high transaction latency.

B. Namecoin

Namecoin [4] first made an attempt to build a domain name service on a blockchain. Anyone can register a domain name on the blockchain by destroying namecoins (0.01 NMC[1]). Destroyed coins can't be used permanently because they are sent to a cryptographic address nobody knows that private key. Theoretically, losing own money in proportion to the number of own registered domain names discourages people from becoming squatters. However, in fact, the value of 0.01 NMC is too low to deter squatters. Kalodne et al. [11] reported that most of the registered domain names on Namecoin are squatted and therefore the Namecoin system is in disrepair.

Additionally, Namecion alleviates front-running by using a two-phase commit process which needs two transactions: a pre-order transaction and a registration transaction. This pre-order method for name registrations is executed as follows.

1) At first, a user sends a pre-order transaction that only broadcasts a name hash.

[1]NMC is the currency code for Namecoin

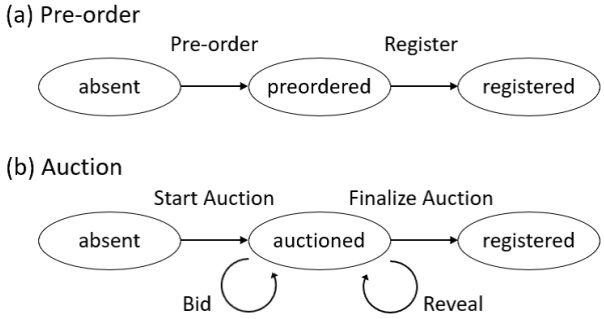

Fig. 1: States and transitions for names when to use the preorder method and the auction method.

2) Next, after the pre-order transaction is confirmed by sequential $m(m \geq 0)$ or more blocks, the same user sends a registration transaction that contains the actual name of that salted hash.

The state changes for names in this process is shown in (a) of Figure 1. In this pre-order method, a front-runner must be delayed at least $m + 1$ blocks. This is because he needs to include his pre-order transaction into one block and wait for m blocks after intercepting the registration transaction. Therefore, the larger m is, the more difficult it is to do front-running.

C. Blockstack

Blockstack [5] is a global naming and storage system secured by Bitcoin, and employs almost the same mechanism as Namecoin to prevent squatting and front-running. One of the main differences between Namecoin and Blockstack is that in the Blockstack system, the amount of coins needed for registering a name is not constant. For instance, the shorter a domain name is, the more coins a registrant needs to destroy for registering it. This is inspired from the observation that shorter names are considered to be more desirable [11]. As a result, Blockstack mitigates squatting more appropriately.

D. Ethereum Name Service (ENS)

ENS [6] is a distributed naming system based on the Ethereum blockchain. ENS uses the auction-based name registration method, which uses a blind auction defined as EIP 162 [13]. Auctions are normally held for 5 days: first 3 days used for bidding and next 2 days used for revealing their bidding money[2]. After this auction period is over, the winning bidder can send a finalizing transaction and register a name by depositing the second highest bidding money[3]. The state changes are shown in (b) of Figure 1. In principle, this auction method prevents both squatting and front-running because the only highest amount bidder can get the target name.

E. Defects of current name registration methods

1) The pre-order method: When using this method, we have to wait for at least the time to mine sequential $m + 1$ blocks.

[2]These periods are determined by the Ethereum timestamp data.

[3]If he is the only bidder, he needs to deposit 0.01 ETH.

TABLE I: ucode structure based on our proposal

Field Name	Length
Version	4 bit
Ucode Allocation Method code (UAMc)	16 bit
Class Code (CC)	4 bit
Owner Level Domain code (OLDc)	n bit
Identification Code (IC)	$(104-n)$ bit

In the case of Namecoin, since $m = 11$ and the mean block time is about 10 minutes, the average waiting time is more than about 120 minutes. In Blockstack community, it is also pointed that some users fail to send a registration transaction because they shutdown their computer before sequential $m+1$ or more blocks is mined [12]. From such characteristics, the pre-order method is not user-friendly.

2) The auction method: Nobody can register a name until the auction period ends. In addition, this method always requires at least three transactions even if the bidder is only one person.

III. Proposed System and Its Parameters

In this section, we briefly explain our proposed ucode ownership management system and implementation parameters about our proposal as a smart contract on Ethereum [10].

A. Proposed ucode structure and representing ownership of allocated ucodes

The current ucode structure [2, 3] was designed to manage ucode ownership by a hierarchical structure. Therefore, we introduce another ucode structure suitable for blockchain-based ucode allocations. Our proposed ucode structure is shown in Table I. The meaning of each field is explained in the following sentences.

Version (Ver):
: This field represents a ucode version number. The current ucode version is "0x0."

Ucode Allocation Method code (UAMc):
: This field, newly defined in this paper, is used for selecting a ucode allocation method (Methods are described later in III-C and III-D). In current ucode management system, this field is used as a TLDc (Top Level Domain code).

Class Code (CC):
: The value of n is determined by the value of the CC (e.g., $n = 8, 24, 40, 56, 72$ or 88); that is, this field determines the boundary between the OLDc and the IC. Both of them are described as follows.

Owner Level Domain Code (OLDc):
: In our system, ucodes are allocated directly to the owner without management organizations. Therefore, current organization identification codes, each of which is called a Second Level Domain code (SLDc), can be replaced as ucode owner identification codes, which are defined as OLD codes. In other words, every ucode's owner is uniquely determined by the first $24+n$ bits of the ucode.

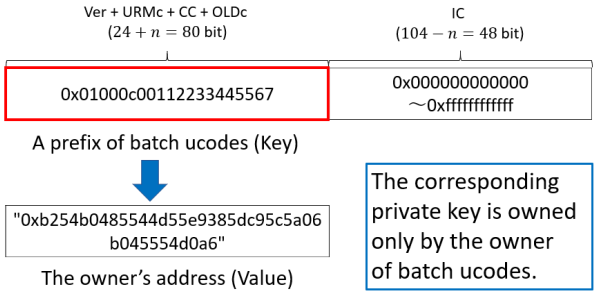

Fig. 2: Mapping the first $24+n$ bits of the ucode to the owner's address identified by his private key (e.g., $n = 56$)

IC (Identification Code):
: This field indicates that the ucode owner identified by the first $24+n$ bits of the ucode can manage the ucode space of the remaining $104-n$ bits.

This ucode structure can be used in the current ucode system by setting every UAMc to a unique value not colliding with the currently used TLD codes. By using this ucode structure, we design a ucode allocation method. The first $24+n$ bits of the ucode is associated with the ucode owner's cryptographic address, as shown in Figure 2. Since each address is uniquely calculated from its corresponding public key, the owner can insist the ownership of allocated ucodes using his digital signature created by his private key.

B. Taking measures against illegally occupying many ucodes

A ucode is a unique numeric identifier which can be issued and used by anyone. On the contrary, it is also important to take measures against particular users or organizations with intention to illegally occupy a lot of ucodes. To deal with such a squatting problem, current blockchain-based naming systems, such as Namecoin, Blockstack and ENS, demand destroying or depositing coins when a user registers a name[4]. This mechanism discourages people to become squatters.

For this reason, our system also requires destroying coins when allocating ucodes, but the amount of coins needed to be destroyed is very low and constant. Two reasons why we can design like that are as follows.

1) The ucode space has 2^{128} or approximately 3.4×10^{38} ucodes. These numerous ucodes make squatting difficult in principle. Therefore, we can lower the price required for ucode allocation.
2) Every ucode is only a unique numeric identifier and doesn't need to be human-readable. Therefore, it is reasonable that the price of each ucode is equal.

C. ucode allocation methods to which current name registration methods are applied

In our system, associating the first $24+n$ bits of the ucode with the owner's address enables us to allocate the remaining $104-n$ bit ucode space to the owner. Now, the important thing is that the ucode prefix can be treated as names because

[4]See section II for further details.

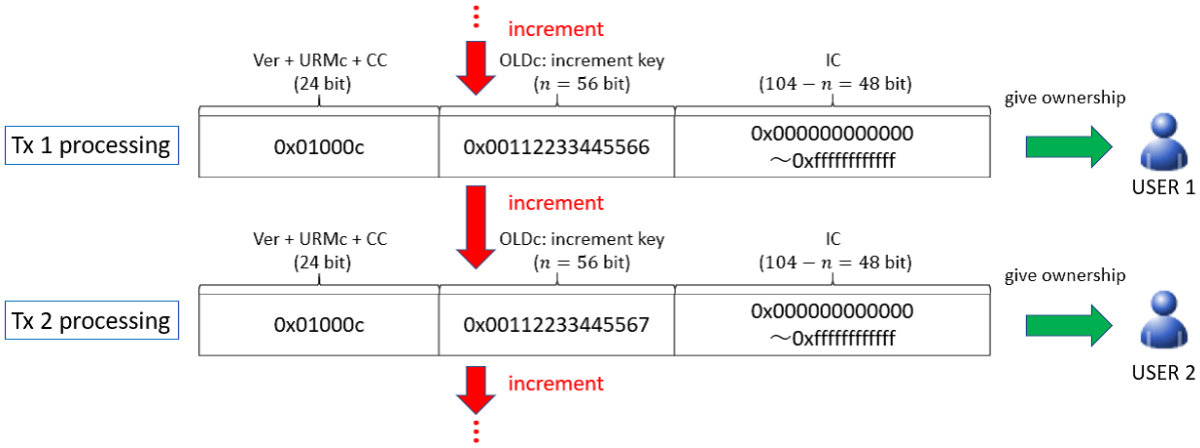

Fig. 3: Sequential ucode allocation by the increment method (e.g., $n = 56$).

characters contain numbers. From this reason, we can easily apply the pre-order name registration method to the Pre-order Ucode Allocation Method (PUAM). We also apply the auction name registration method to the Auction Ucode Allocation Method (AUAM).

D. The increment ucode allocation method (IUAM)

A ucode is only a unique numeric identifier and doesn't have to be human-readable. In other words, users are interested in the size and uniqueness of the allocated ucode space. Therefore, we only focus on guaranteeing that the first 24 + n bits of the ucode is uniquely allocated, and propose the Increment Ucode Allocation Method (IUAM) which assigns sequence numbers to OLD codes automatically. OLD codes, which are treated as increment keys in this method, play a role like AUTO_INCREMENT values in MySQL [18]. The starting value for OLD is 0, and it will increment by 1 at every event that an increment transaction is included in blocks and processed, as shown in Figure 3. Since the first 24 bits are fixed and all increment keys are unique, the allocated ucode space is also unique. The state changes for ucodes in this process is shown in Figure 4.

IUAM is user-friendly and efficient because it requires only one transaction and a simple procedure (All things users have to do is sending an increment transaction). IUAM also prevents both squatting and front-running. Squatters need to send 2^n increment transactions for squatting, but this is impractical given a sufficiently large n due to the need of destroying coins explained in III-B. Front-running is theoretically impossible because all increment transactions always allocate unique ucodes unless the latest increment key overflows. However, instead of these good points, the flexibility of the allocated ucode prefix doesn't exist in IUAM.

E. Transfer ucode ownership to another user

In our system, ucode ownerships can be transferred to another user if he permits the transfer. The reason why the recipient's permission must be needed to change the ownership

Fig. 4: States and transistions for ucodes when to use IUAM

is to protect the recipient from owning malicious ucodes by receiving the ownerships of those ucodes. Therefore, in order to transfer an ownership, the recipient's digital signature on the hash created by the ucode prefix and the expiration timestamp is required. This signature and expiration timestamp are included in the sender's transaction, and it is verified whether they are valid on the blockchain. This mechanism is analogous to the atomic name transfer in [11].

F. Parameters for our implementation on Ethereum

At first, we introduce two simple reasons why we adopt Ethereum for our implementation.

1) Ethereum is one of the most reliable blockchains that can handle digital currency because of its high hash rate.
2) An Ethereum transaction is included into Ethereum blocks faster than a Bitcoin transaction getting into Bitcoin blocks because the current mean block time on Ethereum is about 14.7 seconds[5].

Next, we explain how to set parameters for our proposed system on Ethereum [10].

1) How do we determine the value of n?: The value of n determines that 2^n increment transactions can be executed before the latest increment key overflows. If $n = 24$ and 8 increment transactions are included in every block, increment keys are exhausted in a year when considering the mean block time on Ethereum. Since this situation can easily occur in reality, it is desired that $n = 40, 56, 72$ or 88.

[5]We calculate this value from the Ethereum data between height 5,400,000 and 5,500,000. The mean block time on Bitcoin is about 10 mintutes.

2) How do we determine the price needed for each ucode allocation? : We temporarily set the amount of coins needed to be destoyed per each ucode allocation to 0.0001 ETH[6]. Now, the price of 0.0001 ETH is about 0.06$; that is, squatters need to lose at least 0.06$ per their increment transaction included in blocks. The current total supply of ETH increases approximately linearly over time[7], and the price of 1.0 ETH is changing day by day with the balance of supply and demand. Therefore, in our proposed system, only the ucode allocation price can be changed by authorized organizations' multi-signatures, like ENS root-node management [6].

IV. Simulation for Mean Confirmation Time of Each ucode Allocation Method on Ethereum

In this section, we compute the mean confirmation time of each ucode allocation method. The auction method, which is explained in II-D, is executed based on blockchain timestamps; that is to say, the mean confirmation time of AUAM depends on the system design. Hence, our goal is to get the mean confirmation time of PUAM and IUAM, defined as $E[T_{\mathrm{Pre-order}}]$ and $E[T_{\mathrm{Increment}}]$. To calculate them, let us denote the transaction-confirmation time and the time to mine sequential m blocks as T and S_m, respectively. We also suppose that $S_0 = 0$ and $m \geq 0$. From these definitions, $E[T_{\mathrm{Preorder}}]$ and $E[T_{\mathrm{Increment}}]$ are given by the following equations.

$$\begin{cases} E[T_{\mathrm{Preorder}}] &= E[T] + E[S_m] + E[T] \\ &= E[S_m] + 2E[T]. \\ E[T_{\mathrm{Increment}}] &= E[T]. \end{cases} \quad (1)$$

The above first equation holds because the time to completely execute PUAM is equal to the time to confirm one pre-order transaction, mine sequential m blocks and then confirm one registration transaction[8]. The above second equation also holds because the time to execute IUAM equals the time to confirm one increment transaction. Therefore, our next goal is to get the values of $E[T]$ and $E[S_m]$.

In Bitcoin, the theoretical value of $E(T)$ was derived in [19, eq. (17)], but the author supposed that the block time S_1 is an independent identically distributed exponential variable. However, in Ethereum, this assumption is not justified because the difficulty of mining a block is adjusted in all blocks [10], and block propagation delay [20] is too long to be ignored in comparison with the expected block time. For these reasons, it is difficult to get the theoretical value of $E[T]$ in Ethereum. Therefore, to calculate $E[T]$ numerically, we conduct a simulation by applying the Ethereum block time data into a queuing model which considers the mining process and was proposed in [19].

In addition, by dividing the N pieces of the Ethereum block time data into $K(k_1, k_2, ..., k_{\lfloor N/m \rfloor})$ contiguous sub-periods of length m, we calculate $E[S_m]$ as the element mean of K, as shown in Table II.

TABLE II: Ethereum average time to mine m blocks between height 5,400,000 and 5,500,000.

Number of Blocks	Average Time	Standard Deviation
$m = 10$	146.66 sec	40.71 sec
$m = 20$	293.32 sec	58.68 sec
$m = 30$	439.97 sec	71.30 sec

A. Simulation for calculating mean confirmation time of IUAM and PUAM

To calculate $E[T]$, we perform the same simulation as [19] except that the Ethereum block timestamp data is used as each block creation time.

1) Queuing Model : Transactions arrive at the Ethereum system according to the random Poisson process with rate λ. The maximum number of transactions included in one block is denoted as b. Transactions arriving at the system are served by blocks in a batch manner. When a transaction arrives at the system, the transaction first enters the queue. Even if currently mined block includes smaller than b transactions, the arriving transactions are always included in subsequent blocks and are not served in the current mined block. This transaction behavior details are mentioned in [19].

In Ethereum, b is limited by the amount of consumption, which is called gas [10]. Current each block's gas limit is set to 8,000,000. From the Ethereum data between height 5,400,000 and 5,500,000, the mean gas cost per transaction is 56042, and hence $b = 142$. When $\lambda E[S_1] < b$ is satisfied, the Ethereum blockchain system is stable. Therefore, we assume $\lambda < b/E[S_1] = 9.68$ when we simulate.

By using Ethereum block timestamps between height 5,400,000 and 5,500,000, we conduct a discrete-event simulation whose simulation model is described in IV-A1. Figure 5 shows that substitute the values of $E[T]$ and $E[S_m]$ into equation (1).

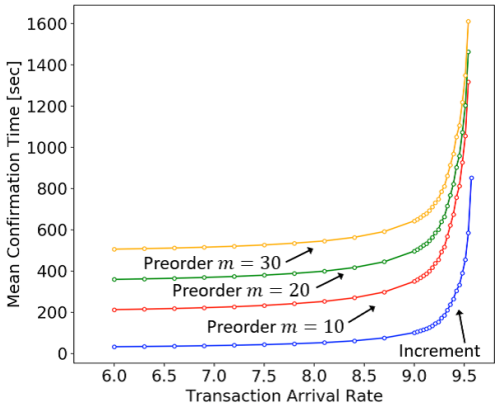

Fig. 5: Increment vs. Preorder: the mean confirmation time from Etehreum data between height 5,400,000 and 5,500,000,

[6]ETH is the currency code for Ethereum

[7]See https://etherscan.io/chart/ethersupplygrowth.

[8]To simplify the blockchain model, we ignore the blockchain reorganization and block propagation delay on the P2P network.

TABLE III: A comparison between the proposed method and conventional methods

	Number of Transactions	Usability	Risk of Front-running	Allocatable ucode Prefix
Pre-order	2	×	At times of congestion	Arbitrary about OLDc
Auction	At least 3	△	None	Arbitrary about OLDc
Increment	1	◯	None	Dependent on the OLDc at the previous ucode allocation

V. Discussion

In this section, we compare each ucode allocation method. Table III summarizes the arguments of V-A∼C.

A. Waiting Time for each ucode allocation mehod

In AUAM, the timings of sending transactions depend on the auction period design. In PUAM, before sending a registration transaction, a user has to wait until at least $m+1$ sequential blocks are mined. From such characteristics, a Blockstack user fails to register his name because he shutdown his PC before its waiting time finishes [12]. On the other hand, in IUAM, a user doesn't have to wait for his increment transaction to be confirmed. He can shutdown his PC at once after sending his increment transaction. This improves IUAM's usability.

B. Risk of front-running when to use PUAM

When we use IUAM and AUAM, it is theoretically impossible to do front-running. On the contrary, when PUAM is used, front-running can occur always. Since a front-runner can pay the highest transaction fee[9], the minimum number of blocks that the front-runner must wait for is $m+2$ (1 block required for confirming the pre-order transaction, m blocks that need to wait for in PUAM, and 1 block required for confirming the registration transaction). Therefore, to prevent front-running, it is desirable that at least the following inequality holds.

$$E[S_{m+2}] < E[T]. \tag{2}$$

The right side of this equation diverges infinitely as $\lambda \to b/E[S_1] = 9.68$; that is to say, if the transactions are congested, front-running can be more easily done. This is a crucial problem in blockchains with low transaction throughput.

C. Flexibility of the allocated ucodes' prefix

AUAM and PUAM can allocate ucodes whose OLD codes are user-defined. However, IUAM can't do this because the value of the OLD code at the ucode allocation is obtained by adding 1 to the value of the OLD code at the previous ucode allocation.

VI. Conclusion

Current ucode ownership management which is maintained by central authorities is cumbersome. Therefore, we designed a blockchain-based ucode ownership management system on a blockchain without a hierarchical structure. Additionally, in this system, we proposed a blockchain-based ucode allocation method, which is called IUAM. Requiring a simple procedure and only one transaction, this method is user-friendly and efficient compared with ucode allocation methods to which we simply apply current name registration methods which require a complex procedure and at least two transactions.

We conducted a case study about our proposed system built on the Ethereum blockchain. We run a queuing simulation for calculating the mean transaction confirmation time on Ethereum. From this result, we revealed that IUAM is superior in terms of the time efficiency, usability and security at the expense of the ucode prefix flexibility.

[9]In blockchains dealing with digital currency, transactions with high transaction fee are likely to be processed quickly.

References

[1] N. Koshizuka and K. Sakamura, "Ubiquitous ID: Standards for Ubiquitous Computing and the Internet of Things," *IEEE Pervasive Computing*, vol. 9, no. 4, pp. 98-101, Oct.-Dec. 2010.

[2] TRON Forum, Ubiquitous ID Center, *ucode: Ubiquitous Code*, 910-S101/UID-00010, Jul. 2009.

[3] ITU-T, *Multimedia information access triggered by tag-based identification*, International Telecommunication Union Recommendation H. 642, Jun. 2012.

[4] Namecoin. https://namecoin.org/.

[5] M. Ali, J. Nelson, R. Shea, and M. J. Freedman, "Blockstack: A Global Naming and Storage System Secured by Blockchains," *USENIX Annual Technical Conference*, p181-194, 2016.

[6] N. Johnson, *ENS Documentation Release 0.1*, https://media.readthedocs.org/pdf/ens/latest/ens.pdf.

[7] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[8] K. Christidis and M. Devetsikiotis, "Blockchains and Smart Contracts for the Internet of Things," *in IEEE Access*, vol. 4, pp. 2292-2303, 2016.

[9] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. E. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, D. Song and R. Wattenhofer, "On Scaling Decentralized Blockchains (A Position Paper)," *In 3rd Workshop on Bitcoin and Blockchain Research*, 2016.

[10] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," Ethereum Project Yellow Paper, vol. 151, 2014.

[11] H. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of Namecoin and lessons for decentralized namespace design," *WEIS ' 15: Proceedings of the 14th Workshop on the Economics of Information Security*, June. 2015.

[12] Blockstack GitHub: issue 1052, *Add more warnings about keeping computer on during name registration*, https://github.com/blockstack/blockstack-browser/issues/1052.

[13] EIP 162, *Initial ENS Registrar Specification*, https://github.com/ethereum/EIPs/issues/162.

[14] J. Gray, "The Transaction Concept: Virtues and Limitations", *Proceedings of the 7th International Conference on Very Large Databases*, pp. 144-154, 1981.

[15] W. Wang, D. T. Hoang, Z. Xiong, D. Niyato, P. Wang, P. Hu and Y. Wen, "A Survey on Consensus Mechanisms and Mining Management in Blockchain Networks," *arXiv:1805.02707v1*, May. 2018.

[16] P. R. Zimmermann. The Official PGP User's Guide. MIT Press, Cambridge, MA, USA, 1995.

[17] ICANN Security and Stability Advisory Committe, *SSAC Advisory on Domain Name Front Running*, SAC 022, Oct. 2007.

[18] MySQL Documentation, *MySQL 5.7 Reference Manual*, https://dev.mysql.com/doc/refman/5.7/en/example-auto-increment.html.

[19] Y. Kawase and S. Kasahara, "Transaction-Confirmation Time for Bitcoin: A Queueing Analytical Approach to Blockchain Mechanism," *Queueing Theory and Network Applications: QTNA 2017*, Lecture Notes in Computer Science, vol. 10591, Nov. 2017.

[20] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," *IEEE P2P 2013 Proceedings*, Trento, 2013, pp. 1-10.