Received 13 September 2017
Revised 21 October 2017
Accepted 21 October 2017

Implementation of blockchain-based energy trading system

Se-Chang Oh
Sejong Cyber University, Seoul, Republic of Korea, and
Min-Soo Kim, Yoon Park, Gyu-Tak Roh and Chin-Woo Lee
Cha University, Gyeonggi-do, Republic of Korea

Abstract

Purpose – The centralized processes of today's power trading systems are complex and pose a risk of price tampering and hacking. The decentralized and unmodifiable nature of the blockchain technology that has recently been highlighted offers the potential to improve this power trading process. The purpose of this study is to implement a system to apply the blockchain technology to the problem of power trading.

Design/methodology/approach – The authors modeled the power trading problem as the interaction between admin, producer and consumer nodes. And a power trading scenario has been created for this model using a blockchain platform called Multichain which is both fast and highly scalable. To verify this scenario, they implemented a trading system using Savoir, a Python-based JsonRPC module.

Findings – Experimental results show that all processes, such as blockchain creation, node connectivity, asset issuance and exchange transactions have been correctly handled according to the scenario.

Originality/value – In this study, the authors have proposed and implemented a power trading method that determines price according to the pure market principle and cannot be manipulated or hacked. It is based on the nature of blockchain technology that is decentralized and cannot be tampered.

Keywords Decentralization, Blockchain, Security, Power trading, Smart contract, Exchange transaction

Paper type Research paper

1. Introduction

The Korean government is seeking ways to revitalize the trading market as part of national energy demand management. At the same time, the government is promoting the proliferation of energy prosumers through the expansion of new and renewable energy supply. Recently, the security issues that have arisen with the introduction of smart grids and the issue of energy suppliers' market manipulation have become a big problem for existing power trading systems. Because it has been revealed that energy suppliers form a cartel to manipulate the price of electricity (Ju-Young, 2015), and that it is easy to hack into the smart grid (Gunhee *et al.*, 2010).

The decentralized nature of the blockchain enables innovation in the transaction process. In other words, it simplifies complex mediation processes and automates all existing

This research was supported by Korea Institute of Building Energy Technology.
This paper forms part of a special section on 'Blockchain on business and entrepreneurship'.

Asia Pacific Journal of Innovation and Entrepreneurship
Vol. 11 No. 3, 2017
pp. 322-334
Emerald Publishing Limited
2071-1395
DOI 10.1108/APJIE-12-2017-037

transactional processes to enable rapid transactions. Also, the nature of the blockchain, which is fundamentally not tamperable, makes the transaction process reliable (Finector research team, 2016).

Thus, the use of blockchains has been studied recently to facilitate energy trading. Overseas examples include a new renewable energy compensation program using Solar Coin (Lucas, 2017), a virtual currency, and Bankymoon to solve power shortage problem with bitcoin donation method using smart meter (KEPCO Economy and Management Research Institute, 2017). In Korea, studies are being conducted to solve the problems of the existing electric power trading system using the blockchain (Lee Chan and Kim Ki, 2017). KEPCO announced plans to build electric vehicle charging stations using blockchain technology and to expand the blockchain platform that combines energy coin (Jae-Sik and Jung-Hyung, 2017). It is also argued that P2P transactions can solve the energy demand forecast failure problem of existing trading systems (Sung-Min, 2017).

This study intends to build a P2P trading platform for household residents who want to buy and sell electric power. Household residents bid and ask with each other on the platform. The platform discloses buyer–sell price information to participants and supports the power trading through smart contracts. In other words, the basic operating process of the platform is designed to automate the transaction process. And approved transaction details are recorded and shared in a distributed ledger. In this process, participants in the network can benefit from securing stability through decentralization and autonomy of pricing.

2. Blockchain-based asset exchange scenario

To enable energy trading using a blockchain, we first need to create a blockchain, connect the producer nodes and the consumer nodes to the generated blockchain, grant producer nodes the authority to issue the energy asset and sell the "ecoin" asset to the consumer nodes so that it can buy energy assets through the blockchain. Once this is done, the producer node can price and sell as much as he wants for his own energy assets. Also, the consumer node can check the sales information and select an appropriate offer to make a transaction. This section shows the process of handling all of these steps using the Command Line Interface (CLI) provided by the blockchain platform named Multichain.

2.1 Creating blockchain

To create a blockchain, run the following command on the terminal (note that "A >" refers to the terminal prompt on the administrator node):

```
A> multichain-util create energybc
```

In this case, the node that created the blockchain has administrator capabilities by default and can implement an exchange site using this feature. Now, to use the generated blockchain, you need to run the daemon for the blockchain service as follows:

```
A> multichaind energybc -daemon
```

This command prints the following message:

```
Other nodes can connect to this node using:
multichaind energybc@172.17.0.1:6823
```

In this case, "energybc@172.17.0.2: 6823" is the address of the administrator node that created the blockchain. Nodes that want to be connected to the generated blockchain need to know this, so they need to be notified through a web page.

All the nodes connected to the blockchain can see the address assigned to the wallet managed by the node through the CLI command "listaddresses" after the blockchain service is started. This address can be used to exchange assets or simple transfer transactions:

```
A> multichain-cli energybc listaddresses
[{
    "address": addr_of_A,
    "ismine": true
}]
```

2.2 Creating assets

By default, administrators have the right to create and issue assets. In this problem, we use two kinds of assets as follows:

(1) "energy": This refers to the electricity generated by the producer. The unit is kW.

(2) "ecoin": This means money to buy electricity. The unit is Korean Republic Won (KRW).

The following command will just generate the asset without issuing it:

```
A> multichain-cli energybc issuefrom addr_of_A addr_of_A
'{"name":"energy","open":true}' 0 0.001 0 '{"origin":"KW",
"stage":"0",
"purpose":"periodic generation"}'
A> multichain-cli energybc issuefrom addr_of_A addr_of_A
'{"name":"ecoin","open":true}' 0 1 0 '{"origin":"KRW",
"stage":"0", "purpose":"ecoin sale"}'
```

2.3 Connection of producer nodes

To connect the producer node to the generated blockchain, the following command should be executed at the site where the producer node will be operated, using the administrator's address "energybc@172.17.0.2: 6823" as a parameter (note that "P >" refers to the terminal prompt on the producer node):

```
P> multichaind energybc@172.17.0.2:6823 -daemon
```

At this time, the following message is displayed. We need to pass the address "addr_of_P", which is initially assigned to the purse of this node, to the administrator node with the information that this node is a producer node. This step should be handled through a web page provided by the administrator node. This is because the producer node is not yet connected to the blockchain.

```
...multichain-cli energybc grant addr_of_P connect,send,receive
```

The administrator node that receives this information connects the producer node to the blockchain through the CLI command as follows:

```
A> multichain-cli energybc grant addr_of_P connect,send,receive
```

The address "addr_of_P" and the fact that this node is a producer node are used in the permission granting step. When the connection is complete, we execute the following command on the producer node.

```
P> multichaind energybc -daemon
```

The blockchain service is started, displaying the address of the node as follows (Figure 1):

```
...multichaind energybc@172.17.0.3:6823
```

2.4 Connection of consumer nodes

Connecting the consumer node to the blockchain is the same as connecting the producer node. The following command should be executed at the site where the consumer node will be operated, using the administrator's address "energybc@172.17.0.2: 6823" as a parameter (note that "C>" refers to the terminal prompt on the consumer node):

```
C> multichaind energybc@172.17.0.2:6823 -daemon
```

At this time, the following message is displayed. We need to pass the address "addr_of_C", which is initially assigned to the purse of this node, to the administrator node with the information that this node is a consumer node. This step should be handled through a web page provided by the administrator node. This is because the consumer node is not yet connected to the blockchain.

```
...multichain-cli energybc grant addr_of_C connect,send,receive
```

The administrator node that receives this information connects the consumer node to the blockchain through the CLI command as follows:

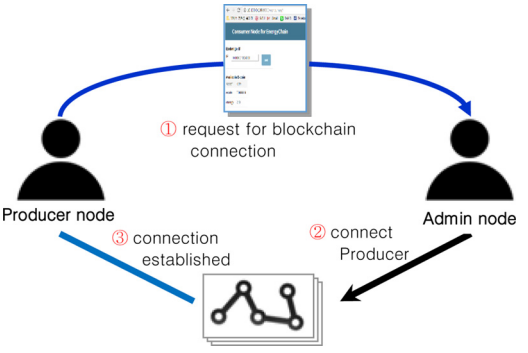

Figure 1.
Process of connecting producer node to blockchain

```
A> multichain-cli energybc grant addr_of_C connect,send,receive
```

When the connection is complete, we execute the following command on the consumer node:

```
C> multichaind energybc -daemon
```

The blockchain service is started, displaying the address of the node as follows:

```
...multichaind energybc@172.17.0.4:6823
```

2.5 Granting producer nodes to reissue "energy" asset

To express the situation in which the producer node generates electricity, it must have the authority to reissue the "energy" asset. However, the producer node connected to the blockchain initially does not have the permission to reissue the asset. Thus, the administrator node needs to grant the "issue" permission for the "energy" asset to all producer nodes. To do this, the administrator node executes the following CLI command:

```
A> multichain-cli energybc grantfrom addr_of_A addr_of_P energy.
  issue
```

The producer node can reissue a certain amount of "energy" assets through the following CLI command:

```
P> multichain-cli energybc issuemore addr_of_P energy 1 0
  '{"origin":"KW","stage":"n","approval":"KEP"}'
```

All assets currently owned by the producer node can be checked through the following CLI command (Figure 2):

```
P> multichain-cli energybc getmultibalances
[{
    "name": "energy",
    "qty": 1.00000000
}]
```

2.6 Purchasing "ecoin" of consumer nodes

The consumer node needs some "ecoin" to purchase the electricity generated by the producer node. Assume that this "ecoin" asset is issued by the administrator node and sold

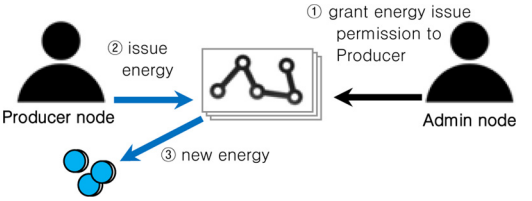

Figure 2.
Process of granting producer node to reissue "energy" asset

to the consumer node. To do this, the consumer node first deposits cash into the manager's bank account. It requests the "ecoin" asset for this via the web service managed by the administrator to send it to the address in the consumer's wallet.

The next CLI command is to send 20,000 units of "ecoin" asset to "addr_of_C", which is the address belonging to the consumer node, for the request of this consumer node.

```
A> multichain-cli energybc issuemore addr_of_C ecoin 20000 0
  '{"origin":"KRW","stage":"n","approval":"KEP"}'
```

The consumer node can check the asset with the following CLI command (Figure 3):

```
C> multichain-cli energybc getmultibalances
[{
   "name": "ecoin",
   "qty": 20000.00000000
}]
```

2.7 Sales of "energy" asset in producer node

The producer node can sell the "energy" asset within the amount of asset that it owns. Therefore, it should first check the amount of the "energy" asset by CLI command as follows:

```
P> multichain-cli energybc getmultibalances
[{
   "name": "energy",
   "qty": 1.00000000
}]
```

As a result, the amount of the "energy" asset is 1 unit, and the producer node requests an exchange transaction to sell one unit of "energy" asset for 1,000 units of the "ecoin" asset through the following CLI command:

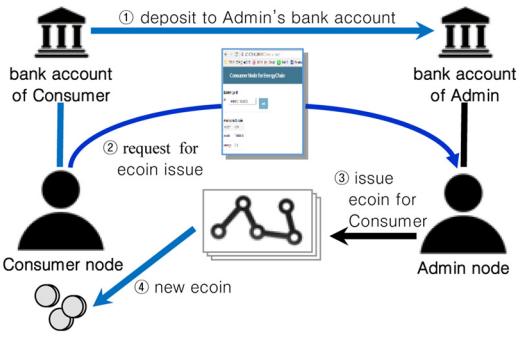

Figure 3.
Process by which consumer node purchase "ecoin" from admin node

```
P> multichain-cli energybc preparelockunspent '{"energy":1}'
  {"txid":a1b2..., "vout":#1}
P> multichain-cli energybc createrawexchange a1b2...#1
  '{"ecoin":1000}'blob1
```

The resulting blob1 includes the contents of the exchange transaction. It is not exposed to other nodes through the blockchain, so it needs to be posted through a stream or a separate web service. Streams cannot modify or delete posted information, and so it is difficult to mark a completed transaction. Therefore, it is appropriate to post the blob1 information through the web service of the exchange operated by the administrator node.

2.8 Purchasing "energy" asset in consumer node

The consumer node uses the following CLI command to check the details of posted blob1:

```
C> multichain-cli energybc decoderawexchange blob1
```

And the contents of the exchange transaction request are displayed as follows:

```
{
    "offer": {
       "amount": 0.00000000,
       "assets": [
          {
               "name": "energy",
               "assetref": "82-266-18690",
               "qty": 1.00000000
          }
       ]
    },
    "ask": {
       "amount": 0.00000000,
       "assets": [
           {
               "name": "ecoin",
               "assetref": "60-265-49207",
               "qty": 1000.00000000
          }
       ]
    },
    ...
    "complete": false
}
```

This means to replace one unit of the "energy" asset with 1,000 units of the "ecoin" asset. If the consumer node agrees with this, it will respond to the exchange transaction. To do this, the consumer node should first check if it has more than 1,000 units of the "ecoin" asset by CLI command as follows:

```
C> multichain-cli energybc gettotalbalances 0
[{
   "name": "ecoin",
   "qty": 10000.00000000
}]
```

As a result, it has more than 1,000 units, so the exchange transaction is completed through three CLI commands as follows:

```
C> multichain-cli energybc preparelockunspentfrom addr_of_C
  '{"ecoin":1000}'{"txid": a3b4..., "vout": #3}
C> multichain-cli energybc appendrawexchange blob1 a3b4... #3
  '{"energy":1}'{"hex": blob2, "complete": true}
C> multichain-cli energybc sendrawtransaction blob2txid
```

The "txid" obtained from the last command is the transaction ID, which means that the transaction has been completed normally. By notifying this "txid" through the web service of the exchange, it can inform the completion of the exchange transaction. The following figure is a flowchart showing whole process of the exchange transaction between the producer node and the consumer node (Figure 4).

3. System implementation

Multichain provides the remote procedure call protocol encoded in JSON (JSON-RPC) application programming interface (API) for applications. Savoir is a module that allows you to use this API based on Python3. Thus, in this study, all functions related to blockchain are implemented using Savoir. It is also necessary to store information in the administrator node to process requests from producer nodes and consumer nodes. To do this, a database is used in the exchange web service managed by the administrator node. As a whole, we need blockchain-related functions, database functions and the user interface. For this purpose, it is appropriate to implement each node as a web service. In this paper, we implemented web service using Django, a Python-based web framework.

The figure below shows the structure of a node that can be implemented as a single physical server or a docker image consisting of a blockchain and a web service. At this node, the user connects to the web server using a web browser, and the web server uses the blockchain function through Savoir (Figure 5).

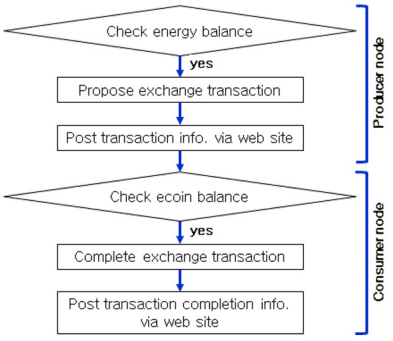

Figure 4.
Process of completing exchange transaction between producer node and consumer node

The following figure shows the overall structure of a blockchain-based energy trading system constructed using the node shown in Figure 5. In this system, all the nodes have the same structure as shown in Figure 5, but the permissions of node and the functions of the web server are different according to their roles (Figure 6).

The following sections describe how the blockchain-related functions are implemented for each type of node using Savoir.

3.1 Implementation of node initialization process

To use blockchain functionality on each node, it is commonly needed to create a Savoir API object and to obtain the address of the wallet stored on each node. These two functions are implemented as follows:

(1) Obtaining the API object: Savoir provides the JSON-RPC API. To use it, you need to get the object in the following way:

```
api = Savoir(rpc_user, rpc_pwd, rpc_ip, rpc_port,
      blockchain_name)
```

Here, each parameter has the following meaning:

- rpc_user: it is set to "multichain" as the ID for obtaining permission to use blockchain via RPC;
- rpc_pwd: the password for RPC, which is determined when each node is connected to the blockchain;

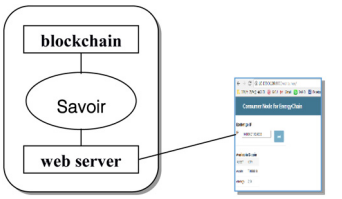

Figure 5.
Structure of node

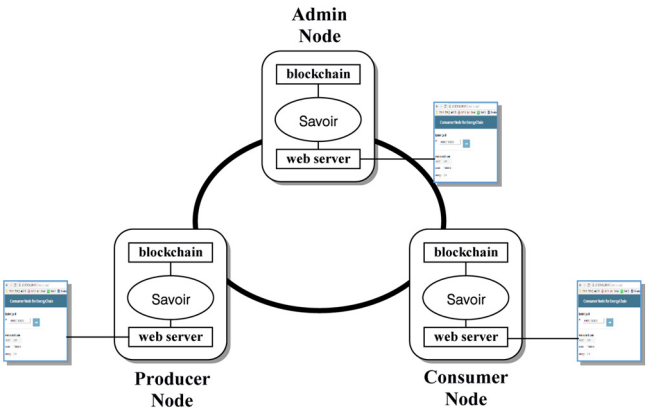

Figure 6.
Structure of blockchain-based energy trading system

- rpc_ip: the Internet Protocol address (IP) of the node where Multichain is installed, which is determined when each node is connected to the blockchain;
- rpc_port: the port assigned to the multichain service, which is determined when initially creating the blockchain; and
- blockchain_name: we can determine when the blockchain is first created and is defined as "energy" in this paper.

(2) Getting addresses: To connect and use the blockchain, both the address of the node itself and the address assigned to the wallet managed by the node are required. These two addresses can be obtained as follows:

- node address: api.getinfo( )["nodeaddress"], and
- wallet address: api.listaddresses( )[0 "address"].

3.2 Implementation of admin node

The following API functions are implemented to handle the requests of the producer node and the consumer node as well as the administrator node:

- connect_node(address, type): This API takes the address, which is the address of the wallet of the requesting node, and the type of the node as parameters and connects them to the blockchain as follows:

```
api.grant(address, "connect,send,receive")
```

If the type is "producer", it grants the permission to reissue "energy" asset as follows:

```
api.grant(addr, "energy.issue")
```

- schedule_ecoin_issue(address, quantity): The consumer transfers the money to the administrator's bank account and requests ecoin issue. This API provides a function to record this request in the DataBase (DB) so that it can be checked later by the administrator.
- issue_ecoin(index): This API provides the function of issuing the ecoin asset to the consumer as follows by checking the consumer's wallet address and issuance amount from DB using the index when the administrator accepts the ecoin issuance request of the consumer.

```
api.issuemore(address, "ecoin", amount)
```

- register_ask(ask_address, blob): This API provides the ability to register a blob in the database. This blob has the contents of the exchange transaction requested by the producer whose address is ask_addres'.
- register_bid(index, bid_address, txid): This API finds a particular exchange transaction in DB using the index. It registers bid_address, which is the address of the consumer, txid and the fact that this transaction is complete to DB.
- exchange_list(status, ask_address, bid_address): This API provides a list of exchange transactions registered in the DB as search results in various ways such as transactions in which the address of producer node is ask_address, transactions

in which the address of consumer node is bid_address, and transactions in a specific state.

3.3 Implementation of producer node

The following API functions are responsible for handling necessary operations according to the request of the producer node.

- reg_exchange_ip(exchange_ip): It provides the function to register the node address of the administrator, who acts as an exchange, into the Producer object. This address is used to propose an exchange transaction or to request a transaction list.
- issue_energy(): It reissues a certain amount of energy asset with the following code:

```
api.issuemore(address, "energy", UNIT_OF_ENERGY_ISSUE)
```

- propose_exchange(qty_energy, qty_ecoin): It provides the function to offer a transaction that exchanges qty_energy units of energy asset with qty_ecoin units of ecoin asset. Where qty_energy must be within its own holdings. To do this, it first checks the asset holdings with the following code:

```
api.getmultibalances()
```

It uses the following code to freeze the assets to be exchanged and creates a transaction:

```
res = api.preparelockunspent ({ASSET_NAME_ENERGY: qty_energy})
blob = api.createrawexchange(res["txid"], res["vout"],
      {ASSET_NAME_MONEY: qty_ecoin})
```

The blob obtained here is sent to the administrator node acting as an exchange and notified.

3.4 Implementation of consumer node

The following API functions are responsible for handling necessary operations according to the request of the consumer node:

- reg_exchange_ip(exchange_ip): It provides the function to register the node address of the administrator, who acts as an exchange, into the Consumer object. This address is used to request of the ecoin issuance to the administrator node, to register a completed exchange transaction or to request a transaction list.
- request_ecoin_issue(qty): It provides the function of requesting the ecoin issuance to the administrator node.
- accept_exchange(index): When the consumer selects a transaction from the list of incomplete exchange transactions, it provides the function to get detailed information on the index and complete the transaction. To do this, it first checks the asset holdings with the following code:

```
api.getmultibalances()
```

It uses the following code to freeze the assets to be exchanged and complete the transaction:

```
res1 = api.preparelockunspent({ASSET_NAME_MONEY: price})
res2 = api.appendrawexchange(blob, res1["txid"], res1["vout"],
       {ASSET_NAME_ENERGY: energy})
```

In this case, res2 ["complete"] will be True when the transaction is complete. The completed transaction is sent to the blockchain using the following code:

```
txid = api.sendrawtransaction(res2["hex"])
```

The finally obtained txid, along with the index of the proposed transaction and the consumer's wallet address, is sent to the administrator node to be registered as the completed transaction.

4. Experimental result

In the experiment, only three nodes were connected to the implemented system. In this case, it took an average of 4.3 s for one transaction to be shared and confirmed among all the nodes. In real situations, such as when applying to residential complexes, hundreds of nodes can be connected. Assuming this situation, the transaction processing time will become a big problem. In addition, Multichain does not support partial transactions where only a portion of the proposed volume is concluded. Therefore, in this study, we implemented only the complete transaction for which the entire volume of proposed transaction is concluded.

5. Conclusion

Today's power trading systems are centralized in their processing, so the price is not determined by the principle of demand and supply, and there is even the risk of price manipulation by collusion. In addition, these systems are vulnerable to security problems such as hacking. Recently, the rapidly developing blockchain technology is inherently decentralized, and is impossible to tamper the recorded information. Applying this to the problem of asset trading simplifies the process and increases the reliability. Therefore, in this study, we implemented an experimental power trading system to demonstrate the possibility that blockchain can be a good solution as suggested in previous studies.

We chose a blockchain platform called Multichain to implement the power trading system. Multichain can define and use two or more assets in a blockchain, so we can trade electricity and money. It is also faster than Bitcoin Core or Ethereum in processing speed and highly scalable.

In this study, we assumed manager, producer and consumer nodes to model the power trading process. The interaction between these three nodes is expressed as a scenario using the CLI command provided by Multichain.

We used a Python-based JsonRPC module called Savoir to implement the trading system. The results are provided in the form of REST APIs that implement the required functionality of the admin, producer and consumer nodes. Experimental results show that all processes such as blockchain generation, connection, asset issuance and transaction are executed correctly according to the scenario.

However, the system implemented in this study is still slow to be applied in real situations where lots of nodes are connected. In addition, Multichain alone is not sufficient for partial transactions where only a portion of the proposed volume is concluded. Therefore, additional efforts are needed to overcome these problems.

References

Finector research team (2016), "Understanding block chains for financial institutions", *Finector Report*, available at: https://drive.google.com/file/d/0B7C70RsMsLrCNFEtNktKLUFCemM/view.

Gunhee, L., Jung-Taek, S. and Cheol-Won, L. (2010), "Status of security promotion for smart grid", *Review of KIISC*, Vol. 20 No. 5, pp. 7-13.

Jae-Sik, G. and Jung-Hyung, C. (2017), *KEPCO Makes World's 1st Blockchain-Based EV Charging Station*, Etnews.

Ju-Young, K. (2015), "Secret manipulation of power trading market", available at: www.labortoday.co.kr/news/articleView.html?idxno=135825, Daily Labor News (accessed 29 December 2015).

KEPCO Economy and Mangagement Research Institute (2017), "Concept and case analysis of blockchain", *KEMRI Power Economy Review*, Vol. 7.

Lee Chan, H. and Kim Ki, H. (2017), "A Study on the Application of Consortium BlockChain to Prosumer Power Trading", *Proceedings of KICIS Conference*, pp. 1441-1442.

Lucas, M. (2017), "Four ways to use Blockchains in your business", CIO Korea, available at: www.ciokorea.com/news/34296#csidxc2a6153358e1001aa6007fc8865a2ab.

Sung-Min, Y. (2017), *P2P Power Trading with Blockchain*, The Science Times.

About the authors

Se-Chang Oh received the MS and PhD degrees in computer science from the KAIST in 1990 and 1997, respectively. He worked at LG Corporation Institute of Technology for four years and worked at Ajou University for three years. Now he is a professor in Computer Software Dept. at Sejong Cyber University. His research interests include blockchain, machine learning and data science. Se-Chang Oh is the corresponding author can be contacted at: scoh713@gmail.com

Min-Soo Kim received the Bachelor's degree in International business from H.U.F.S in 2017. Now he is a CHA business school student at Cha University (CBS). His research interests include blockchain and cryptocurrency.

Yoon Park received the Bachelor's degree in business administration from Sookmyung women's University in 2017. Now she is a CHA business school student at Cha University (CBS). Her research interests include blockchain and cryptocurrency.

Gyu-Tak Roh received the Bachelor's degree in International business from H.U.F.S in 2017. Now he is a CHA business school student at Cha University (CBS). His research interests include blockchain and cryptocurrency.

Chin-Woo Lee received the Bachelor's degree in Business Administration from Texas A&M University-Commerce. Now he is a business school student in Cha University. He researches about the block chain system.