J Inf Process Syst, Vol.16, No.2, pp.301~317, April 2020
https://doi.org/10.3745/JIPS.03.0136

ISSN 1976-913X (Print)
ISSN 2092-805X (Electronic)

JOURNAL OF INFORMATION PROCESSING SYSTEMS JIPS

A Design for a Hyperledger Fabric Blockchain-Based Patch-Management System

Kyoung-Tack Song*, Shee-Ihn Kim*, and Seung-Hee Kim*

Abstract

An enterprise patch-management system (PMS) typically supplies a single point of failure (SPOF) of centralization structure. However, a Blockchain system offers features of decentralization, transaction integrity, user certification, and a smart chaincode. This study proposes a Hyperledger Fabric Blockchain-based distributed patch-management system and verifies its technological feasibility through prototyping, so that all participating users can be protected from various threats. In particular, by adopting a private chain for patch file set management, it is designed as a Blockchain system that can enhance security, log management, latest status supervision and monitoring functions. In addition, it uses a Hyperledger Fabric that owns a practical Byzantine fault tolerant consensus algorithm, and implements the functions of upload patch file set, download patch file set, and audit patch file history, which are major features of PMS, as a smart contract (chaincode), and verified this operation. The distributed ledger structure of Blockchain-based PMS can be a solution for distributor and client authentication and forgery problems, SPOF problem, and distribution record reliability problem. It not only presents an alternative to dealing with central management server loads and failures, but it also provides a higher level of security and availability.

Keywords

Blockchain, Chaincode, Deployment Service, Hyperledger Fabric, Patch

1. Introduction

Recently, methods to effectively utilize the advantages of Blockchain such as confidentiality, integrity, and single point of failure are being studied not only in cryptocurrency but also in other various areas including deep learning [1], notarized documents, trade, secure IoT network [2], and vehicle network [3]. In particular, as these services are universally applied along with the agile methodology based on the web or app to respond to fast and accurate information services in terms of future-oriented usability [4], the importance of reliability and accuracy without forgery for software distribution is increasing.

Patches are usually included in new software releases so that product developers can provide tailored solutions and improved security measures [5]. Even a rudimentary software product may require many updates. All information-technology (IT)-based companies must apply dedicated plans to perform patch-management activities as part of their enterprise management system. With this, an organization-wide

※ This is an Open Access article distributed under the terms of the Creative Commons Attribution Non-Commercial License (http://creativecommons.org/licenses/by-nc/3.0/) which permits unrestricted non-commercial use, distribution, and reproduction in any medium, provided the original work is properly cited.

Manuscript received November 12, 2019; first revision January 31, 2020; accepted February 11, 2020.

Corresponding Author: Seung-Hee Kim (sh.kim@koreatech.ac.kr)

* Dept. of IT Convergence Software Engineering, Korea University of Technology and Education, Cheonan, Korea (skt2000@koreatech.ac.kr, eftpos@koreatech.ac.kr, sh.kim@koreatech.ac.kr)

Copyright© 2020 KIPS

strategy must be established to govern how and when patches are applied to certain systems to manage and mitigate vulnerabilities.

According to a 2017 study on the cost of data leaks, conducted by the Ponemon Institute in England [5], 35% of 2016–2017 cyber-attacks occurred at companies having high-ranking managers who considered cyber-security methods (e.g., patch management) to be of low priority. This implies that companies depending more on IT capabilities not only require higher-level software patch strategies, they require stronger security-minded leadership. However, the important issue here involves understanding how systems having cybersecurity vulnerabilities are updated with usable patches. Most companies do not have an easy time with the patches provided by software suppliers. This is because there is no guarantee that the patch will not conflict with other applications running on the network. Sometimes a balance of priorities is required to minimize interruptions of mission critical systems [6]. Nevertheless, patch management is essential to protect software from cybersecurity threats and to lessen the burdens of system administrators. Patch management requires cognizant support from high-level managers, dedicated resources, clearly defined and assigned responsibilities, creation and maintenance of current technology inventories, vulnerability and patch identification, network scanning and monitoring, pre-distribution test patching, and post-distribution scanning and monitoring [7]. Administration requires timely and practical alerts, patch notices, patch discovery, patch downloads and documentation, vulnerability analysis, prioritization, testing, deployment, and verification and validation [6]. Apart from these dedicated activities, it is necessary to use a system that can overcome security vulnerabilities, protect system features [6], provide automated patching and antivirus protection [8], and ensure the stability of enterprise-wide systems.

This study defines a patch-management system (PMS) as a software system that helps administrators perform company-wide mass management and control of patches and updates for operating systems (OS) and applications on all servers and user computers on a network [9]. Using this definition, we design and verify the technological validity of a Hyperledger Fabric Blockchain-based distributed patch-management system that can reliably protect clients from a variety of threats. To this end, the Blockchain platform uses peer-to-peer (P2P) networking, proof-of-work (PoW) validation, and distributed ledger technology that does not depend on a central server [10] and can transparently and safely manage all transactions. Furthermore, a Blockchain consensus algorithm is used to derive consensus among participants required to attest for the legitimacy of information and to connect patches to the Blockchain. PMS has the problems of SPOF, confidentiality, authenticity, and integrity due to the central server. Furthermore, the Blockchain system is characterized by distributed structure, data tampering prevention, and non-repudiation, therefore, this paper aims to solve the problems of single point of failure (SPOF) by designing a Blockchain-based PMS. This Blockchain architecture, which can manage patches in real time, will contribute greatly to improving the security of patch-management systems and the stability and integrity of patch operations.

The Blockchain theory and preliminary research are examined in Section 2. A design for the Blockchain-based patch-management system is proposed in Section 3. The implementation and experiments are verified in Section 4 using prototypes that implement the proposed design. The results and implications are outlined in Section 5.

2. Preliminary Research

2.1 Patch Management

Previous studies on patch management mainly included works focusing on improving system stability and reliability and on design and implementation of patch-management system improvements (Table 1).

Table 1. Overview of previous studies

Study	Overview of previous studies
Gerace and Cavusoglu [7]	The importance of core elements in patch-management processes
Kim et al. [11]	A method of differentiating the same OS by client and performing various patch deployments
Cavusoglu et al. [12]	The game theoretical model to find a balance between the costs and benefits of patch management and to study the strategic interaction between companies and suppliers
Kim et al.[13]	A web crawler to analyze the structure and characteristics of vendor sites for this purpose
Muhammad and Sinnott [14]	The infrastructure that could overcome problem occurring because of architectural unconditional trust assumptions
Palumbo [15]	The method that leverages a scripting language to resolve problems occurring because of advertisements
Zheng et al. [16]	A method for intrusion protection systems based on virtual machines
Sohn et al. [17]	The model that automated the configuration of databases for security patches from vendor companies
Seo et al. [18]	The system that could download patches from multiple platforms and automatically distribute and install them
Chang et al. [19], Lee et al. [20]	The process that supported heterogeneous environments and used a process cycle and reduced patch-management risks
Kim et al. [21]	The design that used an XML-based system to centrally examine patch installation statuses
Jung et al. [22]	A method using XML to extract network, OSs, hardware, and system information from personal computers (PC)

Gerace and Cavusoglu [7] discussed the importance of core elements in patch-management processes, and Kim et al. [11] proposed a method of differentiating the same OS by client and performing various patch deployments. For time optimization, Cavusoglu et al. [12] developed a game theoretical model to find a balance between the costs and benefits of patch management and to study the strategic interaction between companies and suppliers. Their studies examined the timing of updates and release policies. To implement efficient patch synchronization, they found optimal times and cycles for updates and releases across distributed systems. Kim et al. [13] used a web crawler to analyze the structure and characteristics of vendor sites for this purpose. In Blockchain, the security of collaboration sites is weakened when the dense connection structure between Blockchain services and devices is not safe. Therefore, in a study on improving trustability among nodes having verified stability, Muhammad and Sinnott [14] presented a trust-oriented policy-centered infrastructure that could overcome many of the problems occurring because of architectural unconditional trust assumptions. Additionally, the study expanded the authentication framework for several decision-making entities used to push patches to target nodes.

Studies have also been performed on increasing the stability of patches. Palumbo [15] proposed a method that leverages a scripting language (i.e., PowerShell) rather than a specific system or patch-management method to resolve problems occurring because of advertisements, etc. Zheng et al. [16] proposed a security patch-management method for intrusion protection systems based on virtual machines. Specifically, they proposed a quantitative system security evaluation technique that performed regular vulnerability scans, evaluated system security in terms of availability, and used a composite stochastic reward network (SRN) to detect malicious-attack and system-defense behaviors via an availability-based approach matrix.

Sohn et al. [17] proposed a model that automated the configuration of databases for security patches from vendor companies. Seo et al. [18] proposed a real-time patch-management system that could download patches from multiple platforms and automatically distribute and install them. Chang et al. [19] and Lee et al. [20] proposed a patch-management process that supported heterogeneous environments and used a process cycle and patch strategy that increased the efficiency of enterprise processes and reduced patch-management risks. Extended markup-language (XML)-based security patch-management systems have also been proposed. Kim et al. [21] proposed a patch-management system design that used an XML-based system to centrally examine patch installation statuses of systems and to select appropriate measures when needed. Jung et al. [22] proposed a patch-management system operation method using XML to extract network, OSs, hardware, and system information from personal computers (PC).

As mentioned, there have been no studies on Blockchain-based patch management. Neither have there been any that implemented distributed systems using consensus algorithms beyond XML. Therefore, this study designs a Blockchain-based patch-management system to implement real-time security patches.

2.2 Blockchain Consensus Algorithm

An important part of a Blockchain is the algorithm that creates consensus among participants connected to the network. This is done to verify the legitimacy of the created block of data that is transferred via the Blockchain.

In a typical public Blockchain, a PoW algorithm [23] is used. This is the consensus algorithm that was used in the initial version of the Bitcoin. Ethereum, which handles electronic transactions, used a consensus algorithm that includes both PoW and proof-of-transaction. PoW uses a reverse function to find a certain hash value. As such, it requires advanced computing power.

The proof-of-stake (PoS) algorithm, introduced by Ethereum, was developed to resolve the problem of electricity waste caused by traditional PoW computing. This method found consensus based on assets possessed by nodes. The problem with PoS is that it can be restricted by a certain person, because it grants the discretion to create blocks to the node possessing the stake. To compensate for this problem, the delegated PoS (DPoS) method [24] was developed. The DPoS method is similar to PoS, but it grants the right to create blocks to the top 101 nodes elected by vote. Hyperledger is a private Blockchain, and it has an improved consensus method compared with Bitcoin and Ethereum. Its consensus algorithm uses a process flow that broadly consists of endorsement, orderers, and validations. When it proposes a transaction, the endorsement peer executes the chaincode, and the execution results are signed. The client submits the transaction after collecting the results from the endorsement peers in keeping with the endorsement policy. The orderer service sets the transaction order and creates one block. Before confirming the transaction, each peer verifies whether the endorsement policy conditions have been met

and whether there are any collisions between transactions. By performing all processes normally, consensus is achieved. Practical Byzantine fault tolerance (PBFT) is the main consensus algorithm used in the Hyperledger Fabric Blockchain. It uses a voting mechanism [25], and it is often used in private Blockchains to solve the Byzantine Generals Problem. The algorithm is mostly used in research for financial services (e.g., FinTech). In this study, the PBFT consensus algorithm is used for patch deployment.

2.3 Patch-Management System

The update target-patch file set is configured so that the patch files, which are provided by the maker of the OS or application, are registered with the patch-management system and then deployed and installed on the client. A centralized control patch-management system transmits the patch files or sends a mass command to the installed clients. The clients receive the command and execute the patch file. Normally, the patch-management system is divided into a patch-management service and a deployment service.

As shown in Fig. 1, the patch-management service provides patch-file management, user management, patch-status management, and patch-file verification. The deployment service performs patch-file deployment, patch integrity checks, and patch-file verification. The deployment service includes a patch-status distributed database and a patch-file verification consensus algorithm for verifying patch-file integrity.

The scope of this study includes the portions marked with the red dotted line in Fig. 1. In terms of patch management, the user checks a patch's integrity. With deployment services, the patch file is deployed to the agent via P2P. The patch status for each agent (i.e., whether it was applied normally) is stored in the distributed database of the Blockchain to ensure integrity. The PBFT algorithm, which is the most-used algorithm of private Blockchains, is used as the consensus algorithm for verifying the patch file in the agent.

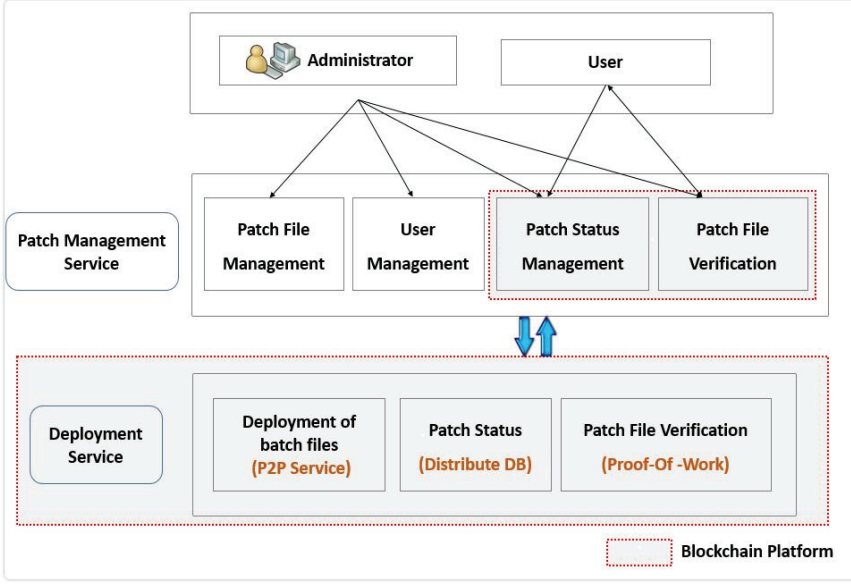

Fig. 1. Architecture of patch-management system.

3. Blockchain-Based Patch-Management System Design

3.1 Principal Concepts of the Patch-Management System

The Blockchain-based patch-management system proposed in this study comprises a patch-management service and a Blockchain platform-based deployment service, as shown in Fig. 2.

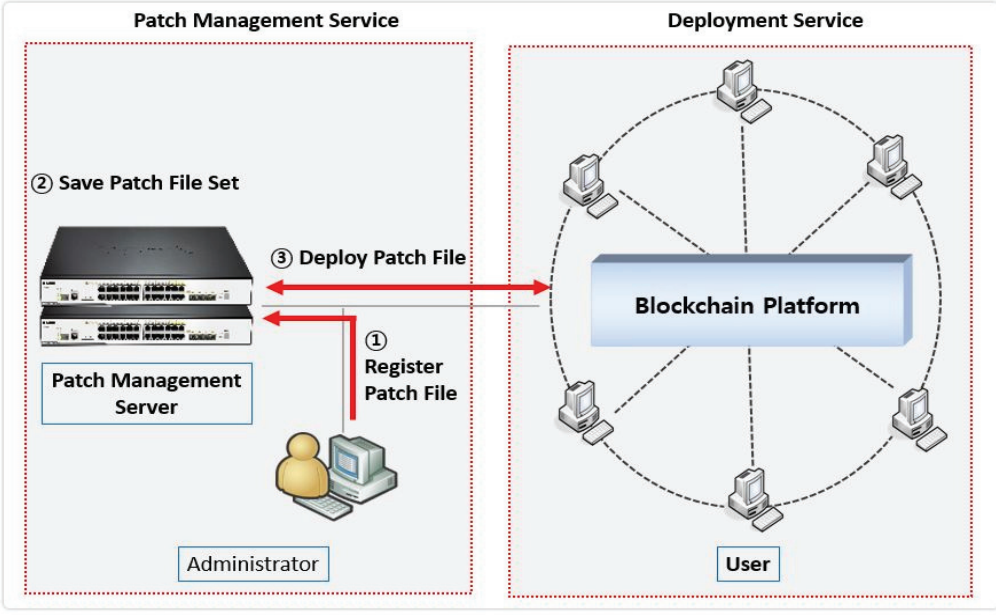

Fig. 2. Conceptual diagram of Blockchain-based patch-management service.

This section examines the service operating process in detail. First, the service manager registers the patch file with the patch-management server. The patch-management server then stores the registered file and sends it to a Blockchain-based deployment service network of service users. The patch-file set sent to the deployment service network is shared to each node (i.e., service user) connected to the Blockchain platform. The detailed process can be examined via the sequence diagram of Fig. 3. The Blockchain-based patch-management system administrator is authenticated, and a registration policy is configured. When service login is performed and the patch file set is registered, the consensus algorithm in the Blockchain service platform automatically creates the Blockchain transaction data that contain the registered patch-file data and deployment policy. The created data are transmitted to the users having joined the server. The users receiving it download the service patch files from the patch-management server, and the files are installed on the users' PCs. Information about who performed the installation is recorded by the deployment service. Then, the administrator monitors the results via the user patch log. This process greatly reduces the server load that would otherwise occur when all users download the patch file from a conventional central-server system. Thus, the patch-management server can be operated in a more stable fashion.

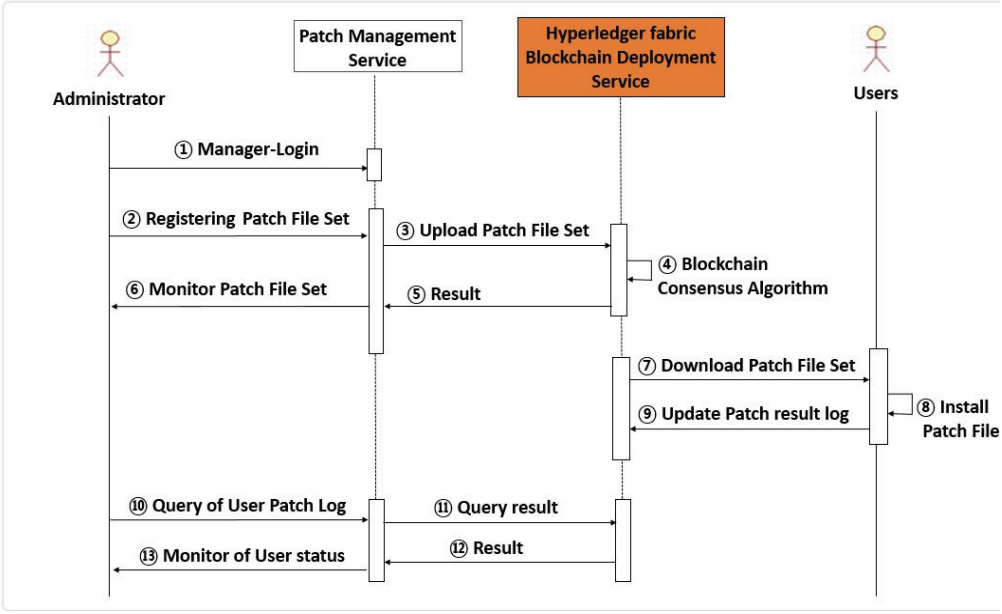

Fig. 3. Sequence diagram for deployment service.

3.2 Blockchain Configuration and Block Structure Design

The problem that needs to be solved in PMS is the SPOF. Furthermore, the log of patching the client PC must be recorded without forgery. The basic functions of the Blockchain system aims to solve this problem. In addition, because the distributor of the patch file must be reliable, it must be composed of the Hyperledger Fabric, which is not a public chain, but a private chain.

Fig. 4 shows the Hyperledger Fabric's Blockchain configuration. There are two nodes (i.e., peers) for each organization (Org1 and Org2). Individual nodes store the same Blockchain distributed ledger. Each organization performs identity verification using a membership service provider (MSP) with certificate authority. Then, a private Blockchain is configured. The Blockchain configured in this manner is assigned to a channel according to usage rights. In this configuration, a single channel is set up. The patch-management server's patch administrator securely uploads the patch file set to be deployed. When the client (user) successfully downloads the patch file set, the results are stored again in the Blockchain.

The Hyperledger Fabric's block structure is shown in Fig. 5. The blocks' header parts are H0, H1, H2, and H3. The header part contains each block's serial number, the previous block's hash value, and the next block's hash value. The blocks' data parts are D0, D1, D2, and D3. They consist of transaction data, which contain information related to the patch file. Finally, the blocks' metadata parts are M0, M1, M2, and M3. They consist of the block's creation time, the block creator's authentication certificate, the public key, and the digital signature for the block. They record whether the transaction is valid or invalid.

The patch file set contains the patch file and related policy information. It is saved as transaction data in the block's data part. Fig. 6 shows the patch-file set, consisting of the patch file name, patch file policy, admin (deployer) identification (ID), and user (client) ID. The policy information of the patch file records whether the file must be used at the client (i.e., mandatory, optional), whether logs are to be left on the

server after the patch file is applied (i.e., write, unwrite), and whether the client is to be rebooted after the updated (i.e., rebooting, unrebooting). The Admin ID is the deployer's ID, and the User ID is the ID of the user (client) who downloads and installs the patch-file set. Transaction data are T1, T2, T3, and T4. Transaction data contain the patch file set which consist of patch file name, patch file policy, admin ID and user ID.

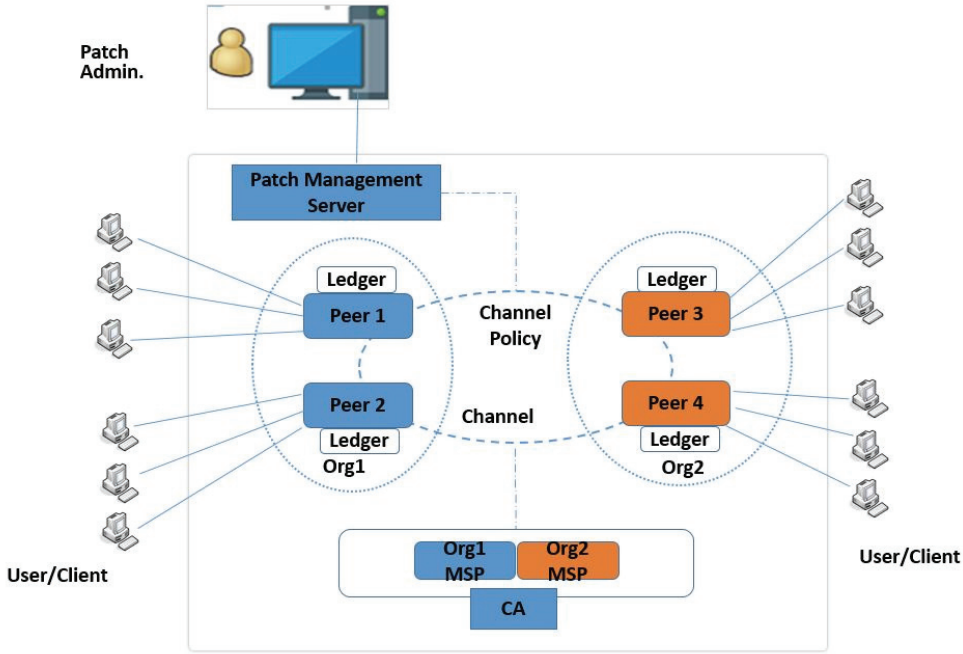

Fig. 4. Blockchain-based deployment service block diagram.

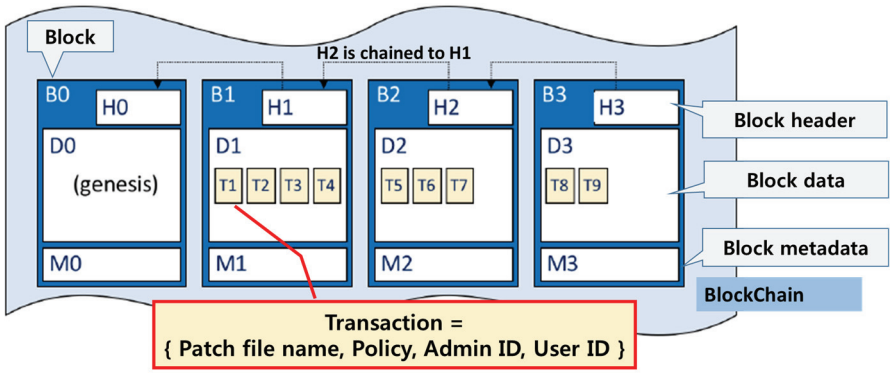

Fig. 5. Block structure of Hyperledger Fabric Blockchain (source from: https://hyperledger-fabric.readthedocs.io).

Fig. 6 shows patch file set in Blockchain. There is "input" area which has chaincode (smart contract) and patch file name, policy information, admin ID and user ID. We will have implemented functions, they are upload function, download function and getHistory function.

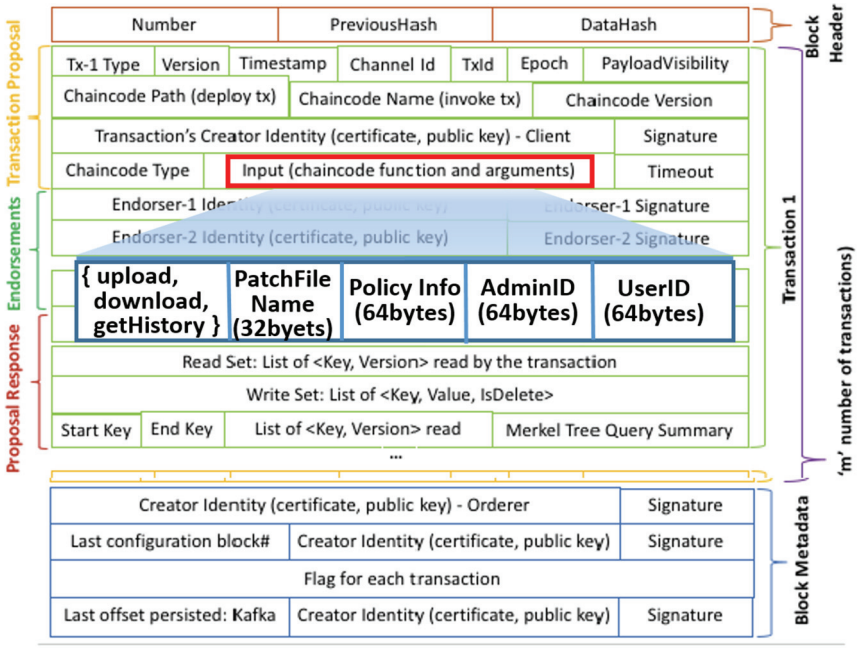

Fig. 6. Patch file set is stored in Blockchain.

4. Implementation and Experiments

4.1 Test Environment

The patch-management system comprises a private Blockchain platform. For this study, it consists of one peer in Org1 with one channel, as shown in Fig. 7. Because peers have the ledger, they store the patch-file set and the log data regarding downloads. Peers also contain the rights-managing MSP, the chaincode, the Blockchain ledger, and the world-state database. Peers create a Blockchain platform based

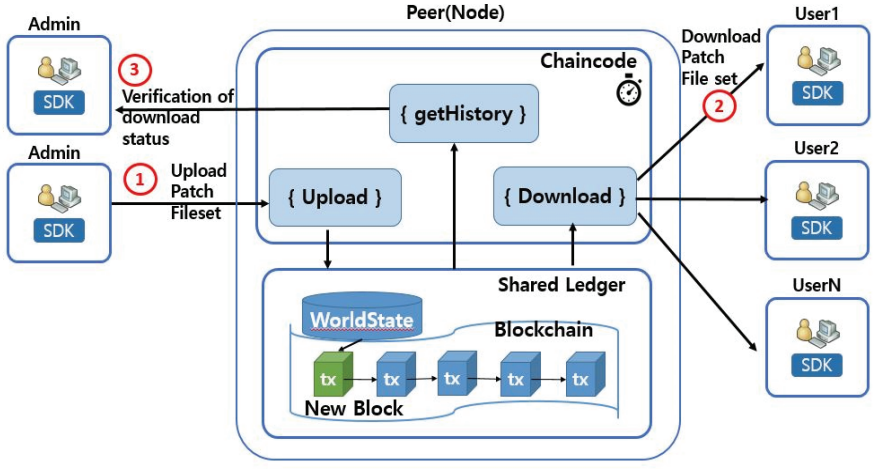

Fig. 7. Hyperledger Fabric experiment environment diagram.

on Ubuntu, CouchDB, Hyperledger 1.4.0, and a Docker, which is a Linux container technology. They also use the Go language for patch-file registration and deployment services. The Chrome plug-in program, Restlet, is used for the admin and client environment. In the peers (nodes), Apache Tomcat is used so that the Restful application programming interface can be used to execute the chaincode. The admin uploads the patch file set, the client downloads the patch file set, and logs are recorded. After this, the admin checks to see which client applied the patch-file set.

4.2 Experiment Scenario

The main functions of the PMS are uploading the patch files to the server, downloading the patch files to the client/user side, and recording the results. The experimental process is shown in Fig. 8. The experiment is divided into three parts: uploading the patch file set, downloading the patch files and recording logs, and download verification. The source file for the environment settings of The Hyperledger Fabric Blockchain platform was created as a YAML data serialization file. The main environment settings file is shown in Table 2. In Exam 1, admin saves a set of patch files in the distributed ledger of the Hyperledger Fabric. In Exam 2, the user downloads a set of patch files from the distributed ledger, records the fact that the files were downloaded. In Exam 3, admin verifies the fact that the user has downloaded. In other words, since the user downloads a patch file set from any distributed node, the SPOF problem that is generated by the existence of a central server only can be solved.

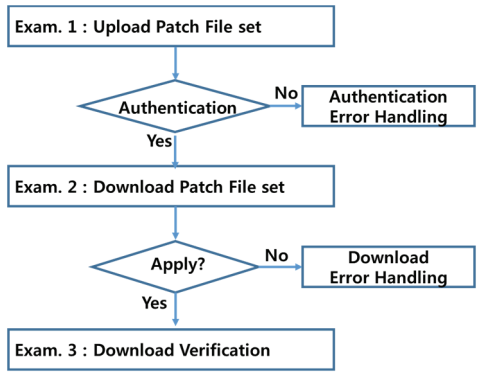

Fig. 8. Test scenario flow.

Table 2. Configuration file

Filename	Description
configtx.yaml	Hyperledger network configuration settings
crypto-config.yaml	Organization and participants' certificate issuance settings and issuance definitions
docker-compose-cli.yaml	Network node and docker settings
patchcfilecc.go	Chain code related to patch file set (upload, download, verification)

4.3 Experiment Results

Figs. 9–11 are parts of the source code used for registering and deploying the patch-file set. These are chaincodes created for the patch-file deployment service. The patch-file set data registered by the administrator include all of these data, used as chaincode in the Blockchain system. A connection between

the node and the channel is required to execute the features defined by the chaincode. The mutual connection is performed internally by the Hyperledger Fabric system. That is, the chaincode has three forms: one used by the admin to upload the patch-file set to the block chain and another used by the user to download the patch-file set, recording a log of the download in the Blockchain, and the other used by the admin to verify the installation of patch file set by users. Fig. 9 displays the main function of the patch-management chaincode for this experiment. Via this function, experiments were performed on the patch-file set upload, the patch-file set download and log recording, and download verification.

```
// Patch Management System's chaincode(smart contract)

// initialization & import module

// define "Patch File Set "
type patchfileset struct {
    PatchFileName string `json:"filename"` // Patch file
    PolicyInfo    string `json:"Info"`     // Patch file policy
    AdminID       string `json:adminID`    // Uploader ID
    UserID        string `json:userID`     // Downloader ID
}

// Main Function for the experiment of this paper
func main() {
    err := shim.Start(new(PatchChaincode))
}

// Init - initialize chaincode
func (t *PatchChaincode) Init(stub shim.ChaincodeStubInterface) pb.Response { return shim.Success(nil) }

// Invoke - Our entry point for chaincode(smart contract)
func (t *PatchChaincode) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
    function, args := stub.GetFunctionAndParameters()
    // Chaincode execution by admin or user request
    if function == "upload" {  return t.upload(stub, args) }            // Upload patch file set
    else if function == "download" { return t.download(stub, args) }    // Download patch file set
    else if function == "getHistory" {return t.getHistory(stub, args)}  // Audit patch file set
}
```

Fig. 9. Patch-management chaincode (main function).

```
// upload - upload patch file set by Admin
func (t *PatchChaincode) upload(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    // ==== Input sanitization ====
    // Error if input parameter is not 3

    // Assigning the passed parameter as a variable to be used for the blockchain
    patchfileName := "Patch File Name"
    policyInfo := "Rebooting or Not Rebooting After Patch"
    adminID := "Admin ID"

    // ==== Check if patch file set already exists ====
    patchfileAsBytes, err := stub.GetState(patchfileName)

    // define patchfileset for admin's upload
    patchfileset := &patchfileset{ patchfileName, policyInfo, adminID, "null"}

    // ==== Create patch file set object and marshal to JSON ==
    patchfileJSONasBytes, err := json.Marshal(patchfileset)

    // === Save patch file set to state DB ===
    err = stub.PutState(patchfileName, patchfileJSONasBytes)
    return shim.Success(nil)
}
```

Fig. 10. Patch-management chaincode (admin, patch file-set upload).

4.3.1. Uploading patch file sets

The administrator uploading the patch-file set to the patch-management system is the deployer. Fig. 10 shows the upload function used when the deployer uploads a patch-file set. Information about the

blockchain about the patch-file name, patch-file policy information, and deployer ID must be recorded. Registering the patch-file set uses the Hyperledger Fabric Blockchain's own function to ensure the integrity of the deployer's (admin's) execution record. For the deployer to deploy the patch file set, it must first be registered in the Hyperledger Fabric's Blockchain. To do this, the upload function is called with functional arguments that include the patch-file name, patch-file policy information, and deployer ID (AdminID). Fig. 12 is a screen shot that shows the patch file-set being uploaded and its resultant values.

```
// download - download patch file set and write log by User
func (t *PatchChaincode) download(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    // ==== Input sanitization ====
    // Error if input the number of parameter is not 2
    // Assign value to variables which are saved at blockchain

    // ==== Check if patch file set already exists ====
    valAsbytes, err := stub.GetState(patchfileName)

    patchfilesetDown := patchfileset{}
    json.Unmarshal(valAsbytes, &patchfilesetDown)

    // Assign the userID who download patch file set
    patchfilesetDown.UserID = userId

    // ==== Create patch file set object and marshal to JSON ====
    patchfilesetJSONasBytes, err := json.Marshal(patchfilesetDown)
    stub.PutState(patchfileName, patchfilesetJSONasBytes)

    return shim.Success(valAsbytes)
}
```

Fig. 11. Patch-management chaincode (user, download/log record).

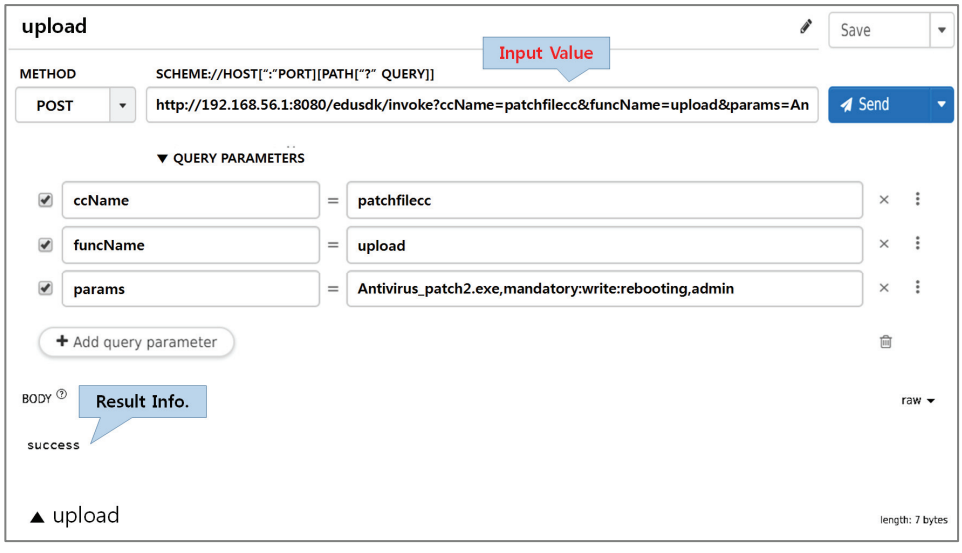

Fig. 12. Patch management chaincode (admin, patch file-set upload) execution results.

4.3.2. Downloading patch file sets and recording logs

The patch-file set is downloaded from the patch-management system, and the user who installs the patch file on the client PC is the client who uses the patch-management system. The patch files recorded

in the Blockchain are used, and data on the patch file set include the patch-file name, patch-file policy data, and deployer ID. The patch-file name is searched, and the patch-file policy data and deployer ID are downloaded. To leave a patch log, the user ID is recorded in the Blockchain, which can be used to prove that the client safely applied the patching.

Fig. 11 is the chaincode with which the client (user) downloads the patch file set and records a log. Fig. 13 is the chaincode execution screen and the log recording results screen.

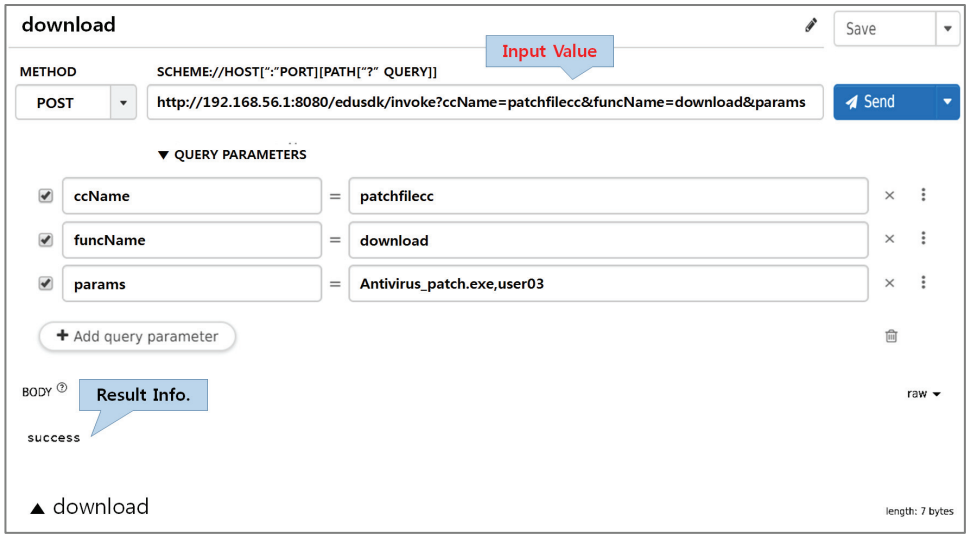

Fig. 13. Patch-management chaincode (user, download/log record) results.

4.3.3. Patch file set's download log check (Audit)

In the patch-management system, it is necessary to verify that the client downloaded the patch-file set and successfully installed it. These log records are proof of downloading and are recorded without tampering with the Blockchain. To check the download record, the admin reads the transaction information in the Blockchain, as shown in Fig. 14. Fig. 15 is a screen-shot that shows the getHistory() execution results.

```
// getHistory - get transaction history from blockchain by Admin
func (t *PatchChaincode) getHistory(stub shim.ChaincodeStubInterface, args []string) pb.Response {
    // ==== Input sanitation ====
    // Error if input the number of parameter is not 1

    // Assign value to variable which is used for search
    patchfileName := "Patch File Name"

    // Get the upload and download list of patch file set
    resultsIterator, err := stub.GetHistoryForKey(patchfileName)

    // Get the result from GetHistoryForKey() Function
    for resultsIterator.HasNext() {
        response, err := resultsIterator.Next()
        // Get the search result from results and make strings
    }
    return shim.Success(buffer.Bytes())
}
```

Fig. 14. Patch-management chaincode (admin, audit).

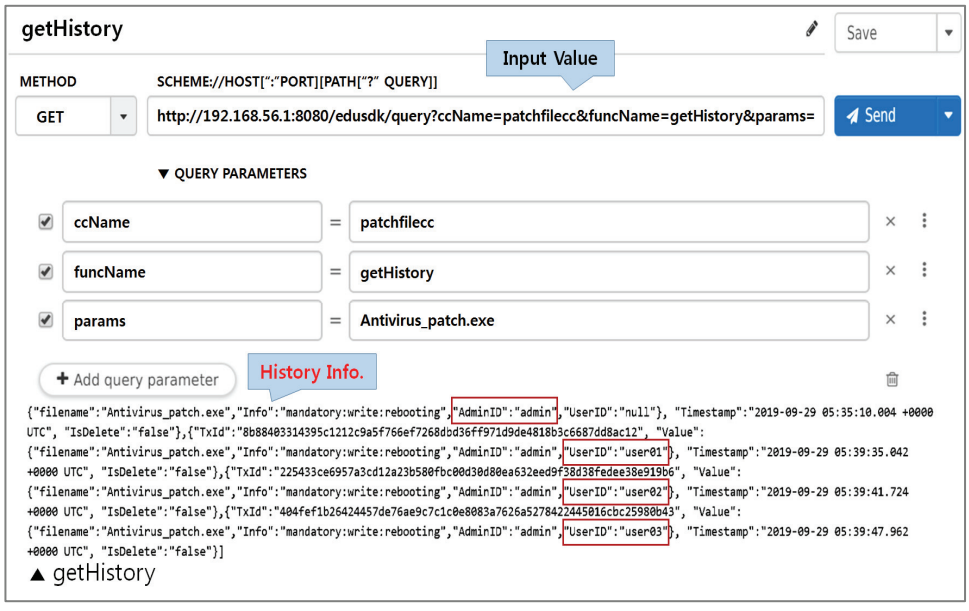

Fig. 15. Patch-management chaincode (admin, audit) results.

4.3.4. Implications

We tested the feasibility of implementing a Blockchain-based patch-management system by confirming that the patch-file set admin uploaded the set to the block chain, that the user downloaded the patch-file set, and that the log record was used to verify downloading. By using Hyperledger Fabric Blockchain technology, the patch-file management approach takes advantage of the benefits of endorsing, ordering, and validating the admin's patch-file set registration transaction at different nodes (peers).

By doing so, In general, a patch is a method of exchanging information between lines of a document file for each code and exchanging information through a file described in a standard diff format. Therefore, various diff formats exist and this has been the biggest cause of errors when applying patch files. However, patches have been evolving such that only the modified file information can be managed even when there are patches that are not confirmed due to a complicated source or when changes need to be applied by applying a different patch file.

The patch file management that employs the Hyperledger Fabric Blockchain technology can solve the errors caused by the diff technology (description format) for information exchange while using the Blockchain consensus algorithm as it is. Moreover, it can provide a technical alternative that can increase reliability the ease of use and reduce the risk of patch application.

Integrity can be assured, tampering can be prevented, and undeniable signatures will exist in the patch-file set. Therefore, patch-management systems that use Hyperledger Fabric Blockchain technology are highly significant, not only because they resolve the single-point-of-failure problem, but that they also greatly increase security, stability, and availability without a complex system implementation.

Based on these results, the use of two-factor authentication for administrators controlling patch-file registration and deployment policies directly strengthen security. It is also possible to consider a design in which the deployer and the administrator are different people. Another approach to strengthening

integrity verification is to provide private and public keys to each deployer, to have the deployer use a private key to sign the patch-file set when it is registered, and to add the public key to the transaction.

5. Conclusion

For stable patch management, it is necessary to use a system that provides security vulnerability resolution, system feature protection, automated security patch management, and antivirus protection to ensure the stability of an enterprise system. This study proposed a Hyperledger Fabric Blockchain-based distributed patch-management system and verified its technological feasibility via prototyping, enabling all participants to be protected from various deployment threats. In the deployment service, patch files were deployed to the agent via P2P. The Blockchain's distributed database storage method and a PBFT consensus algorithm technique were used to verify that the patches were executed normally. By doing so, the integrity of the patch application status database was verified.

Whereas this study verified the technical feasibility of Blockchain technology-based patch management, follow-up studies should be conducted based on the structure proposed in this study. They should seek to strengthen integrity and control structural priorities with two-factor authentication with various deployers and administrators. A patch-management approach that uses this study's Hyperledger Fabric Blockchain technology may not only fundamentally resolve the problems of centralized server loads and failures, it may also contribute greatly toward improving the integrity of patch services and the operability and availability of patch-management systems.

Acknowledgement

This paper is created using research funds from the 2018 New Professor Research Project of Korea University of Technology and Education.

References

[1] S. Rathore, Y. Pan, and J. H. Park, "BlockDeepNet: a Blockchain-based secure deep learning for IoT network," *Sustainability*, vol. 11, no.14, article no. 3974, 2019.

[2] S. Rathore, B. W. Kwon, and J. H. Park, "BlockSecIoTNet: Blockchain-based decentralized security architecture for IoT network," *Journal of Network and Computer Applications*, vol. 143, pp. 167-177, 2019.

[3] P. K. Sharma, S. Y. Moon, and J. H. Park, "Block-VN: a distributed Blockchain based vehicular network architecture in smart city," *Journal of Information Processing Systems*, vol. 13, no. 1, pp. 184-195, 2017.

[4] Y. Sung and J. H. Park, "Future trends of Blockchain and crypto currency: challenges, opportunities, and solutions," *Journal of Information Processing Systems*, vol. 15, no. 3, pp. 457-463, 2019.

[5] C. Dennis, "Why is patch management necessary?," *Network Security*, vol. 2018, no.7, pp. 9-13, 2018.

[6] S. Liu, R. Kuhn, and H. Rossman, "Surviving insecure it: effective patch management," *IT Professional*, vol. 11, no. 2, pp. 49-51, 2009.

[7] T. Gerace and H. Cavusoglu, "The critical elements of patch management," in *Proceedings of the 33rd Annual ACM SIGUCCS Conference on User services*, Monterey, CA, 2005, pp. 98-101.

[8] C. Higby and M. Bailey "Wireless security patch management system," in *Proceedings of the 5th Conference on Information Technology Education*, Salt Lake City, UT, 2004, pp. 165-168.

[9] I. Y. Lee, S. Y. Lee, J. S. Moon, and J. I. Lim, "A study on efficient component in patch management system," in *Proceedings of the Korean Society of Broad Engineers Conference*, 2008, pp. 21-24.

[10] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," 2008; https://git.dhimmel.com/bitcoin-whitepaper/.

[11] Y. J. Kim, S. W. Lee, T. S. Sohn, J. S. Moon, J. T. Seo, J. B. Yun, and E. K. Park, "Design the classed patch distribution system framework considering the extension," in *Proceedings of the Korean Institute of Information Science Society Conference*, 2004, pp. 199-201.

[12] H. Cavusoglu, H. Cavusoglu, and J. Zhang, "Security patch management: Share the burden or share the damage?," *Management Science*, vol. 54, no. 4, pp. 657-670, 2008.

[13] Y. Kim, S. Na, H. Kim, and Y. Won, "Automatic patch information collection system using web crawler," *Journal of The Korea Institute of Information Security and Cryptology*, vol. 28, no. 6, pp. 1393-1399, 2018.

[14] J. Muhammad and R. O. Sinnott, "Policy-driven patch management for distributed environments," in *Proceedings of 2009 3rd International Conference on Network and System Security*, Gold Coast, Australia, 2009, pp. 158-163.

[15] T. Palumbo, "The power of PowerShell: examples of how PowerShell scripts can supplement a patch management system to solve unusual problems," in *Proceedings of the 2017 ACM Annual Conference on SIGUCCS*, Seattle, WA, 2017, pp. 7-14.

[16] J. Zheng, H. Okamura, and T. Dohi, "Security evaluation of a VM-based intrusion-tolerant system with pull-type patch management," in *Proceedings of 2019 IEEE 19th International Symposium on High Assurance Systems Engineering (HASE)*, Hangzhou, China, 2019, pp. 156-163.

[17] T. S. Sohn, J. W. Kim, I. G. Park, J. S. Moon, J. T. Seo, E. K. Im, and C. W. Lee, "The design of a secure patch distribution architecture," in *Proceedings of the Korean Information Science Society Conference*, 2002, pp. 559-561.

[18] J. T. Seo, J. B. Yun, D. S. Choi, E. K. Park, J. W. Seo, T. S. Sohn, and J. S. Moon, "Patch management system with multiplatform support," in *Proceedings of the Korean Information Science Society Conference*, 2003, pp. 889-891.

[19] C. W. Chang, D. R. Tsai, and J. M. Tsai, "A cross-site patch management model and architecture design for large scale heterogeneous environment," in *Proceedings of the 39th Annual 2005 International Carnahan Conference on Security Technology*, Las Palmas, Spain, 2005, pp. 41-46.

[20] S. W. Lee, Y. J. Kim, T. S. Sohn, J. S. Moon, J. T. Seo, E. Y. Lee, and D. H. Lee, "Design the normalized secure patch distribution & management system," in *Proceedings of the Korean Information Science Society Conference*, 2004, pp. 502-504.

[21] Y. J. Kim, S. W. Lee, T. S. Sohn, J. S. Moon, J. T. Seo, E. Y. Lee, and E. K. Park, "Design the security patch central management system using XML," in *Proceedings of the Korean Information Science Society Conference*, 2004, pp. 505-507.

[22] I. T. Jung, T. H. Han, and H. S. Jo, "PC information extraction method applicable to MacOS-based patch management system," in *Proceedings of the Korean Information Science Society Conference*, 2017, pp. 1536-1538.

[23] A. M. Antonopoulos, *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. Sebastopol, CA: O'Reilly Media Inc., 2014.

[24] D. Larimer, "Delegated Proof-of-Stake (DPoS)," 2014; https://bitcointalk.org/index.php?topic=558316.0.

[25] G. T. Nguyen and K. Kim, "A survey about consensus algorithms used in Blockchain," *Journal of Information Processing Systems*, vol. 14, no.1, pp. 101-128, 2018.

Kyoung-Tack Song **https://orcid.org/0000-0001-5530-7948**

He received a bachelor's degree at Dankook University. Currently, he is studying in IT convergence SW engineering department of Korea University of Technology and Education of Korea. The major interest is personal information security using blockchain.

Shee-Ihn Kim **https://orcid.org/0000-0002-9909-1795**

He received a bachelor's degree at Korea University. Currently, he is studying in IT convergence SW engineering department of Korea University of Technology and Education. The major interest areas consist of blockchain, payment gateway and local exchange trading system.

Seung-Hee Kim **https://orcid.org/0000-0001-6312-9846**

She received a bachelor's degree at Dongguk University, a master's degree at Yonsei University, and doctor's degrees in Industry Information System of IT Policy School at Seoul National University of Science & Technology of Korea. Currently, she is an assistant professor of Korea University of technology and education. The major interest areas consist of Blockchain, software quality engineering and optimization.