

BlockSubPay - A Blockchain Framework for Subscription-Based Payment in Cloud Service

Yustus Eko Oktian, Elizabeth Nathania Witanto, Sandra Kumi, Sang-Gon Lee*

Department of Computer Engineering, Dongseo University

yustus.oktian@gmail.com, elizabethnathaniaw93@gmail.com, kumisandra54@gmail.com, nok60@dongseo.ac.kr

Abstract— The payment of a subscription-based model for the cloud service is regularly operated with the association of the bank as a centralized trustful third party. This system taxes the parties involved by requiring them to pay high transaction costs to the bank. In this paper, we propose a protocol to enable *Fixed Subscription* and *Pay As You Go Subscription* payments in the cloud service without the use of the third party. We achieve this goal by decentralizing the payment with blockchain. In return, this decentralized system generates low transaction costs. We implement our idea in the form of two smart contracts in Ethereum network. We also evaluate the security properties of our protocol. Based on the performance analysis, we can argue that all of the operations in the smart contract produces reasonable gas costs and the contracts are usable in the real network.

Keywords— blockchain, smart contract, subscription payment, cloud service

I. INTRODUCTION

Generally, the cloud providers operate the cloud computing business with two economic approaches: free and paid service. They allow free services to users with expectations that they can analyze users' data for advertising. Alternatively, the providers can also sell the cloud services in a paid subscription-based model. To access the cloud resource, the client has to subscribe to the service by paying a subscription fee on a term basis (e.g. every month, every three months, or every year).

We can divide two types of paid subscription-based model in cloud services:

- **Fixed Subscription.** The services that charge clients a fixed amount of subscription fee in every period. For instance, Google One requires about \$10 per month for a 2 TB storage [1].
- **Pay As You Go Subscription.** The cloud services can generate different amount of bill per term based on the clients' usage. For example, The price for Amazon Web Service (AWS) Machine Learning as a Service (MLaS) servers to compute is \$0.42 per hour and the cost of 1000 prediction results is \$0.10 [2]. The bill increase along the increase of usage.

A well-known business model is the cloud providers offer several types of subscription-based payments to choose. The

TABLE 1. COMPARISON OF SEVERAL PAYMENT METHODS

Method	Time	Cost
Check	3-7 days	\$15-20
Credit/Debit Card	2-3 days	\$0.25 & 2-3%
ACH	2-3 days	\$1.96
Wire Transfer	1-5 days	\$35-75
Bitcoin	10-120 minutes	\$3-7
Ethereum	0.5-2 minutes	\$0.007-0.1

clients subscribe to one of them according to their requirements. The clients then pay the subscription fee with the help of the trustful centralized bank. Most of the payments are made through Visa and MasterCard. The payments flow through three backend intermediaries: the Issuer (i.e. the bank that is used by the clients), the Acquirer (i.e. the bank used by the cloud providers), and the Switch (i.e. the intermediary that glue all of those banks together). Each of them taxes the money being transferred as transaction fees. The higher the money being transacted the higher the cut [3].

The rise of cryptocurrency in the most recent years can help to reduce the transaction cost. Bitcoin [4] can achieve relatively low transaction cost when a high amount of money being transferred. Ethereum [5] in the other hand always generate low transaction cost regardless of the volume of money. Furthermore, both of them can deliver the money in minutes instead of days as depicted in Table 1 [3].

In this paper, we propose a subscription-based payment for cloud services using the smart contract as an escrow between the clients and the cloud providers. Specifically, the smart contract [6] will displace the role of the centralized bank. In result, we can build a subscription-based payment that is more transparent and fair with low transaction cost.

We made the following contributions in this paper:

- We design a protocol that enables the clients to subscribe to the cloud providers using *Fixed Subscription* and *Pay As You Go Subscription* method without the need of the trustful centralized third party.
- We implement the protocol in the Ethereum blockchain network and ensure the usability and feasibility of the protocol by presenting rigorous unit tests.
- We provide the security analysis of our protocol as well as performance measurements of our implementation.

*Corresponding Author

Roadmaps. We organize the rest of the paper as follows. Section 2 describes our protocol design of *Fixed Subscription* and *Pay As You Go Subscription*. Section 3 explains our implementation method. Section 4 details the evaluation of our proposal regarding the security properties and performance analysis. We present the related works in Section 5. Lastly, we conclude in Section 6.

II. PROPOSED PROTOCOL

In this section, we outline our proposed protocol to allow payments from clients to cloud providers for both *Fixed Subscription* and *Pay As You Go Subscription*. The main goal of our protocol design is to ensure that the associated party (i.e. the client and the cloud provider) cannot cheat the subscription process. They need to keep being honest so that no one feels at disadvantaged.

The proposed protocol comprises four stages: (i) Subscription Registration Stage, (ii) Access Token Request Stage, (iii) Cloud Resource Access Stage, and (iv) Subscription Expiration Stage.

A. Subscription Registration Stage

This first step contains all of the setups required to initiate the subscription from the clients. Figure 1 depicts the overall operations in this stage.

(i) The client notifies the cloud provider that he requires to get access to the cloud resources. This step is off-chain, meaning that it is not happening in the blockchain network. Therefore, the client can determine the availability of the cloud provider using any discovery method. For example, the provider can deploy a web app that acts as a gateway to initiate the protocol. Afterward, the client can ask for cloud resources by accessing the provider's web page.

(ii) The provider acknowledges the client's request by sending the response message back to the client. It contains all of the required information to start a subscription service to this provider. For instance, it holds the smart contract address to which the client has to make the payment. It also carries the rate of the subscription, which is the number of amounts the client has to pay to the smart contract.

(iii) The client calls a 'subscribe' method in the smart contract to begin the subscription registration. By calling this method, the client also pays the subscription fee to the smart contract.

(iv) The smart contract verifies the payments whether it includes the right amounts. Upon successful confirmation, the smart contract records the public key address used to transfer the payment into the pool list of public keys. All of the public keys stored in this list are eligible to access the servers since they are subscribers who already paid the fee. Furthermore, note that the smart contract does not directly forward the funds that it received to the provider. Instead, the provider can claim the money later at the end of the subscription period. This delay is necessary to guarantee the fairness of the protocol.

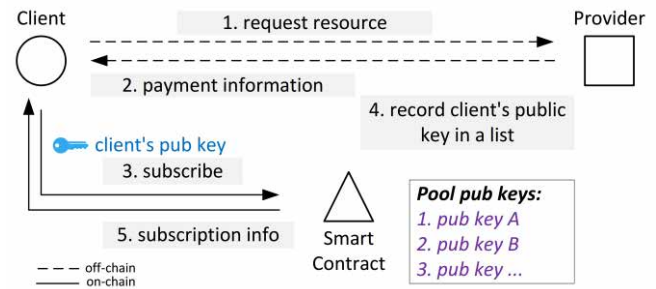


Figure 1. Subscription Registration Stage

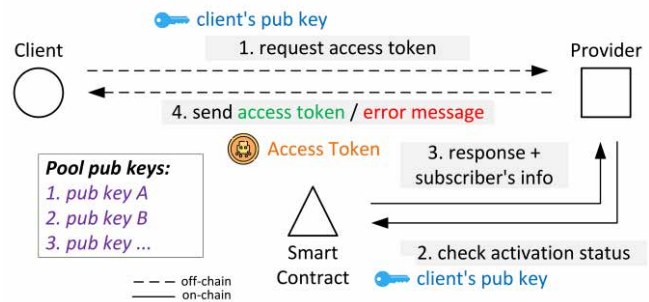


Figure 2. Access Token Request Stage

(v) Finally, the smart contract returns the subscription information such as subscription status and expiry date to the client. For *Pay As You Go Subscription*, the smart contract also informs the current client's usage, which is the current usage at the server. This value should be zero because the client does not utilize any processing power at the server yet. Alternatively, the provider can also send an error message at this step if he found that the payment made in the previous step is invalid.

Subscription fee. The smart contract can determine the amount of the subscription fee that the client has to pay as follows:

- **Fixed Subscription.** The client pays the specific amount of the subscription rate according to the one that he is interested in. For instance, if the client subscribes to a \$10 per month subscription, he needs to spend exactly \$10 in this stage.
- **Pay As You Go Subscription.** Since the cloud provider charges the client based on the client's usage later, the client does not understand how much money he should pay at this point. The best-case scenario is that the client is honest and he adjusts the payment according to the usage that he planned to use. Let says that the client needs to use the servers for 10 hours and the rate of usage per hour is \$0.5. It implies that the client has to pay \$5 to satisfy his requirements. To mitigate rogue subscribers, the provider can set a minimum deposit policy. In this approach, the client has to put some deposits and place their money at stake with an intention to make him become honest.

B. Access Token Request Stage

The second stage details the access token request from the client to the provider. The access token is a symbolic portrayal of the qualification to enter the servers in the cloud provider. Those who own the tokens is authorized to retrieve or process the resource in the server. The main idea in this stage is that the client who had paid the subscription fee is eligible to get the access token from the provider. The provider can implement any access token technologies such as Bearer Token [7] or JSON Web Token (JWT) [8]. Figure 2 depicts the overall procedure in this stage.

(i) First, the client initiates this stage by asking an access token to the provider. The cloud provider can open a gateway for the client to request for access tokens. For instance, the provider can expose REST API endpoints to the client. Afterward, the client signs a request with its public key. The client has to use the same public key with the one that he applied during the payment for the subscription fee in the previous stage.

(ii) In the following step, the provider checks the eligibility of the client with its supplied public key. The provider calls a method in the smart contract to check for the subscription status of the given public key. The provider gives the access token only if the client has an active subscription status. If the smart contract cannot find the public key, it indicates that the client is not a subscriber or has not paid the subscription fee yet. Thus, the provider can mark the client as invalid.

(iii) The smart contract transmits the results of the public key verification back to the cloud provider. It also supplies the provider with details of subscriber's information such as expiry date, current client's usage, and the number of deposits.

(iv) The cloud provider sends access token to the client when the verification is successful. Otherwise, it returns error messages detailing the missing requirements.

Subscription status. The smart contract determines whether a client has an active subscription or not as follows:

- **Fixed Subscription.** A subscription will remain active as long as it is not expired.
- **Pay As You Go Subscription.** The smart contract set the client active if it still has deposits left in the accounts. Specifically, the deposits need to be higher than the expected bills based on the current client's usages. For instance, a client has \$10 deposits. The rate of the subscription fee is \$0.5 per hour. Then, if the client has 10 hours usage, the projected bills become \$5 (i.e. the \$0.5 rate times 10 hours). In this case, the client's status is still active. However, the client becomes inactive once he reaches 20-hour usages. Lastly, the subscription status will also become inactive when it is expired.

C. Cloud Resource Access Stage

The third stage operates when the client access the cloud provider's resources. The main goal in this stage is that only authenticated and authorized client is able to enter the servers. Figure 3 depicts all the operations in this stage.

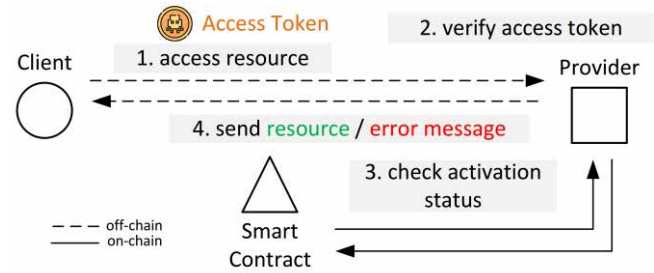


Figure 3. Cloud Resource Access Stage

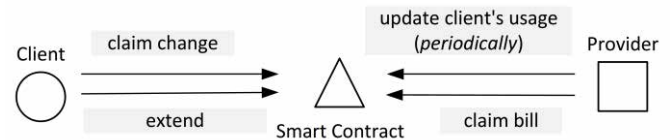


Figure 4. Subscription Expiration Stage

(i) The client calls the cloud provider's endpoints to access the resource while also presenting its access token. For instance, the client uses the REST API from the provider to access the resource.

(ii) The provider receives the client's request and verifies the authenticity and the authority of the given access token. Specifically, the provider needs to check if the access token is expired. Furthermore, the access token needs to have the right privilege to access the server. Depending on the implementation, some tokens may have higher-level access privilege than the others.

(iii) The provider also needs to check the subscription status of the client when the client enters the server. The provider should only allow the entrance of an access token tied to the subscribers that have an active subscription status. Otherwise, the provider rejects the request and revoke the access token. The provider can check for the status of the client by calling the method in the smart contract similar to one in the previous stage.

(iv) When the provider finds out that the token is valid, he allows the request continues to be processed in the servers and transmits the response back to the client according to the inquired tasks. Otherwise, it drops the request and returns the error messages.

D. Subscription Expiration Stage

The client and the provider enter this final stage when the subscription status of the client becomes inactive. Several scenarios can occur in this stage. The steps for each of them can also be different depending on the subscription type. Figure 4 depicts all of the possible plots.

Subscription Extension. During this expiration stage, the access token of the client will be inoperable thus the client cannot reach the resource anymore. To maintain the use of the servers in the cloud provider, the client has to resolve the subscription status first.

- *Fixed Subscription.* The client can prolong the subscription by calling the ‘extend subscription’ method in the smart contract. Consequently, the client also has to spend the required subscription fee to the smart contract. The client can stay subscribing to the same subscription plan as before. Otherwise, he can also decide to downgrade or upgrade the subscription by selecting the lower or higher tier variant. Once the smart contract confirms the payment, the subscribers will have new active subscription status along with an updated expiry date.
- *Pay As You Go Subscription.* To extend the subscription, the client simply needs to recharge his deposits in the smart contract. Once the smart contract verifies the deposit transfers, the expiration date of the client will be prolonged along with an updated deposit balance.

Subscription Fee Withdrawal. During the registration stage, the client paid the subscription fee to the smart contract. In other words, the smart contract holds the aggregated payments from multiple clients temporarily. The cloud provider can withdraw this money by calling a ‘claim fee’ method in the smart contract. We can argue that this approach is the most important features in the smart contract. Therefore, we need to make sure that only the provider can withdraw the money.

- *Fixed Subscription.* The amount of subscription fees that the provider can gather depends on the number of clients subscribe to the resource. It also depends on the subscription tier that the clients pick.
- *Pay As You Go Subscription.* The amount of subscription fees that the provider can collect depends on not only the number of subscribers but also the usage of the client in the provider’s resource. To facilitate the calculation for the usage-based charge, the provider requires to present proofs of client’s usage to the smart contract. Specifically, the provider has to transmit the usage report periodically (e.g. every hour) to the smart contract. Based on this report, the smart contract can estimate the bill that the cloud provider can withdraw.

Subscription Cancellation. When the client loses interest in any particular subscription, he can choose to cancel the subscription with no additional cost.

- *Fixed Subscription.* The client simply does nothing during this stage to cancel the subscription. The subscription status of the client remains inactive until the client starts the payment to begin another subscription period. While inactive, the client cannot access the server and the provider cannot charge money from the client.
- *Pay As You Go Subscription.* Similar to the *Fixed Subscription*, the client requires no steps to cancel the subscription. However, since the user placed some deposits to the smart contract during the registration period, we need to provide a method for the client to withdraw the deposits from the smart contract. The amount of the deposit that can be withdrawn depends on

the client’s usage in the server. If the client has not used any resource in the server yet, it is possible to withdraw the full amount of the deposits. If the client uses 10 hours from a \$0.5 per hour usage rate, the generated bill that the provider can claim is \$5. Assuming the original deposit made by the client was \$10, the amount of deposit that the client can withdraw will be \$5.

III. IMPLEMENTATION

We create two smart contracts, one to implement the *Fixed Subscription* and another one to realize the *Pay As You Go Subscription*. We code our smart contract in Solidity programming language [9]. Then, we develop the smart contract in Ethereum blockchain [5]. During our smart contract development phase, we deploy our smart contract in a local network using Ganache [10] instead of using the real Ethereum network. However, both networks should generate identical outcomes. Lastly, we test our smart contract using the Truffle framework [11].

IV. EVALUATION

In this section, we describe the evaluation of our proposed system. First, we analyze the security properties of our working protocol. Second, we measure the performance of our smart contract and the cost to do some transactions using our system.

A. Trust Issues

Because we introduce the decentralization in our system, there is no trusted third party to govern the subscription process. This absent raise the trust issues between the client and the provider. We detail several scenarios in the following paragraph.

Scenario #1. The provider can discriminate the client by not serving the client even though he already paid the deposit or the subscription fee. The malicious intention here is to make the client lose the money he already spent on the fee.

Solution. In *Pay As You Go Subscription*, the client can get his deposit back at the fourth stage when the subscription status is inactive. If the client is blocked from the server, the client’s usage will not be updated. Therefore, the client can retrieve a fair amount of deposits back. Unfortunately, we do not have a similar refund policy for the *Fixed Subscription* in our current implementation. However, we will add this feature in our future work. Moreover, we have to mention that the client subscribes to the cloud provider using its public key as its pseudonym. When the provider victimizes the client by banning his public key, the client can simply create a new public key to bypass this censorship.

Scenario #2. The client can gain benefits over the provider by spending a small number of deposits but trying to consume more hours usage than the deposits allotted. For example, assume that the provider charges the client \$0.5 per hour for the server usage. The client intentionally deposits \$2 even

though he already arranged to use the server for more than 5 hours.

Solution. In our protocol, the provider updates the client's usage regularly. When the provider recognizes that the client reaches its limit (i.e 4 hours usage based on the \$2 deposit), it revokes the access token and blocks further requests from this client. The more frequent the provider update the usage, the more precise the smart contract can detect the abuse of resource usage.

Scenario #3. The client gets advantages over the provider by quickly refunding the deposits in the smart contract before the provider is able to update the client's usage. After paying the subscription fee, assume that the client uses the server for 30 minutes. Let us also consider that the provider refreshes the client's usage every hour. If the client can refund the deposit before the provider updates the usage at the first hour, the client earns free 30 minutes access.

Solution. In our implementation, we made a simple adjustment such that the client can only redeem the deposit when the subscription status is inactive. With this time limit window, the provider can have enough chance to update the client's usage to eliminate this issue.

Scenario #4. The provider can give a fake report usage to gain advantages from the client. For instance, the provider deliberately reports a false 1000 hours usage even though the client only consumes 10 hours usage.

Solution. We can mitigate this issue by implementing a credible usage report system on the server side. We can integrate the system with the smart contract that we have developed. Therefore, we consider this solution to be our future work.

B. Integrity

It is critical that our smart contract must have a bug-free property. Therefore, we create several unit tests in our program as our first action to ensure that the smart contract that we develop is working as intended. The test includes multiple scenarios ranging from the creation of new subscriptions to the withdrawal of bills and deposit refunds. In the current implementation, we count there are 68 unit tests in total.

C. Performance

One of the methods to rank the performance of a smart contract is to examine the gas used during the operations of the smart contract. The gas usage depends on the complexity of the smart contract. Generally, the more complicated the program is the more gas is required to run the operation. The storage procedure call is the most expensive one. Therefore, write operation produces higher gas than the read operation.

We measure and record the gas usage of our smart contract methods in Table 2 for *Fixed Subscription* and Table 3 for *Pay As You Go Subscription*. We omit the read-only methods from the tables since they generate a small number of gas. Several things worth mentioning from the tables are as follows. First,

TABLE 2. FIXED SUBSCRIPTION EVALUATION

Initiator	Method	Gas	Cost
Provider	Update Current Rate	43716	\$0.02
Provider	Claim Bill	20762	\$0.01
Client	Extend Subscription	34383	\$0.016
Client	New Subscription	109240	\$0.051
Provider	Contract Creation	1101327	\$0.512

TABLE 3. PAY AS YOU GO SUBSCRIPTION EVALUATION

Initiator	Method	Gas	Cost
Provider	Update Minimum Usage	43508	\$0.02
Provider	Update Current Rate	43826	\$0.02
Provider	Update Subscriber Usage	51098	\$0.024
Provider	Claim Bill	35883	\$0.017
Client	Refund Deposit	38771	\$0.018
Client	Recharge Deposit	34859	\$0.016
Client	New Subscription	134931	\$0.063
Provider	Contract Creation	2532423	\$1.178

the contract creation is the process that consumes the highest gas among all operations in both tables. Fortunately, this contract creation occurs once for all. Thus, we can argue that the future revenue can easily compensate the high gas cost. Second, we can observe that the operations to create new subscriptions generate high gas usage. We also expect this case to happen since we have to store subscribers' data during this operation. However, the client runs this operation once so it is not a big issue. The thing that we put many considerations here is the method to update the client usage. The provider has to call this method periodically to be able to charge the user. The more frequent the provider updates the usage, the more precise the bill will be. However, the provider needs to consider the gas cost as it can potentially make the provider lose some of his profits from the bill if the update rate is very high.

Finally, we put the estimation of the transaction price in Table 2 and 3. We notice that most of the operations in our smart contract are very cheap for both the client and the provider. We retrieve the price from the ETH Gas Station website [12]. It is a reliable service to foresee the outcome of a transaction before the user actually transmits the transaction to the Ethereum blockchain network. We choose the standard price with the gas price of 3 Gwei at the time of our writing (Jan 3, 2019). With this gas price, we can confirm that the miners will record the transaction in the blockchain in the next 5 minutes.

V. RELATED WORK

BCPay [13] is a blockchain based fair payment framework that is utilized to outsource services in cloud computing. Having the same goal with ours, this proposal also explores and examines the feasibility and security properties of payments without the third-party relationship. Based on their evaluation, the protocol can achieve soundness and robust fairness while producing an adequate number of transactions and computation cost.

Secure Pub-Sub [14] is another blockchain-based fair payment with the reputation system for publish and subscribe systems. The publishers can issue a topic in the blockchain. The subscribers, which are interested in the published topic, can make a request of the issue by giving payments in deposits. Afterward, the publisher encrypts the topics and transmits them to the subscribers. Eventually, the subscribers can decrypt them and rate the publishers with a good or bad reputation.

To the best of our understanding, both proposals are motivated by the Bitcoin [4]. Thus, they present heavy scripts full of cryptographic proofs as extensions to the Bitcoin. Meanwhile, we take the Ethereum blockchain [5] approach by accomplishing similar goals with the smart contract. To better understand our differentiation, we have to recall that both Bitcoin and Ethereum are blockchain technologies however they are entirely distinct in their implementations [15].

Another proposal that is closely related to ours can be found in Saranyu [16]. Their study adopts the Ethereum smart contract and attempts to tackle the payments of cloud services. They present the idea of delegating cloud resource management with smart contracts. Their method holds four services: identity management, authentication, authorization, and charging (i.e. the payments). Unfortunately, the authors do not specify any detail concerning the design and implementation of the payments in their paper.

VI. CONCLUSION

In this paper, we presented our proposed protocol to enable *Fixed Subscription* and *Pay As You Go Subscription* payments in the cloud service without the use of the third party. We implemented our idea in a form of two smart contracts in Ethereum network. We have outlined the design of our protocol and evaluated the security properties. Based on the performance measurement, we can conclude that all of the operations in the smart contract generates reasonable gas and produce low transaction costs. For our future work, we plan to add new features such as refund policy and credible client's usage reporting. We also plan to assess the scalability of our protocol in a seamlessly integrated environment between the authentication, authorization, accounting, and payments.

ACKNOWLEDGMENT

This work was supported by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (Grant Number: 2018R1D1A1B07047601).

REFERENCES

- [1] Google. (2018, Dec 14). *One simple way to get more out of Google* [Online]. Available: <https://one.google.com/about>.
- [2] A. W. Service. (2018, Dec 14). *Amazon Machine Learning Pricing* [Online]. Available: <https://aws.amazon.com/aml/pricing/>.
- [3] K. Kreder. (2018, Dec 14). *Money vs. Cryptocurrency, The Real Costs (part 1)* [Online]. Available: <https://blog.gridplus.io/money-vs-cryptocurrency-the-real-costs-part-1-33c09dfea671>.
- [4] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.
- [5] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, pp. 1-32, 2014.
- [6] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, 1997.
- [7] M. Jones and D. Hardt, "The OAuth 2.0 authorization framework: Bearer token usage," 2070-1721, 2012.
- [8] M. Jones et al., "JSON Web Token (JWT)," 2012.
- [9] (2018, Dec 14). *Solidity* [Online]. Available: <https://solidity.readthedocs.io/en/v0.4.24/>.
- [10] T. Suite. (2018, Dec 14). *Ganache. One Click Blockchain* [Online]. Available: <https://truffleframework.com/ganache>.
- [11] T. Suite. (2018, Dec 14). *Truffle. Smart Contract Made Sweeter* [Online]. Available: <https://truffleframework.com/truffle>.
- [12] (2018, Nov 26). *ETH Gas Station* [Online]. Available: <https://ethgasstation.info/index.php>.
- [13] Y. Zhang, R. H. Deng, X. Liu, and D. Zheng, "Blockchain based Efficient and Robust Fair Payment for Outsourcing Services in Cloud Computing," *Information Sciences*, 2018.
- [14] Y. Zhao, Y. Li, Q. Mu, B. Yang, and Y. Yu, "Secure Pub-Sub: Blockchain-Based Fair Payment With Reputation for Reliable Cyber Physical Systems," *IEEE Access*, vol. 6, pp. 12295-12303, 2018.
- [15] V. Saini. (2018, Nov 26). *Getting Deep Into Ethereum: How Data Is Stored In Ethereum?* [Online]. Available: <https://hackernoon.com/getting-deep-into-ethereum-how-data-is-stored-in-ethereum-e3f669d96033>.
- [16] S. Nayak, N. C. Narendra, A. Shukla, and J. Kempf, "Saranyu: Using Smart Contracts and Blockchain for Cloud Tenant Management," in *2018 IEEE 11th International Conference on Cloud Computing (CLOUD)*, 2018: IEEE, pp. 857-861.



Yustus Eko Oktian is a Ph.D. student at Dongseo University, South Korea. He received his bachelor degree in Electrical Engineering from Petra Christian University, Indonesia in 2013 and his master degree in Computer Engineering from Dongseo University, South Korea in 2016. His research interests are on the topics of network security, distributed computing, Blockchain, Internet-of-Things (IoT), and Software Defined Networking (SDN).



Elizabeth N. Witanto received her BCS degree from the Department of Informatics Engineering, Petra Christian University, Indonesia in 2015. She is expected to complete her M.Sc. from Graduate School of Computer Engineering, Dongseo University, South Korea in August 2020. Her research interests are focused on Internet of Things (IoT), Blockchain.



Sandra Kumi received a B.S. degree in Computer Science from Kwame Nkrumah University of Science and Technology, Ghana in 2016. Currently, she is a Master student in the Department of Computer Engineering, Dongseo University, South Korea and expected to graduate in the year 2020. Her research interests are computer and network security and software defined networking.



Sang-Gon Lee received his BEng, MEng, and PhD degrees in electronics engineering from Kyungpook National University, Rep of Korea, in 1986, 1988, and 1993, respectively. He is a professor in the Division of Computer Engineering, Dongseo University, Busan, Rep. of Korea. He was a visiting scholar at QUT, Australia, from August 2003 to July 2004 and at the University of Alabama at Huntsville, USA, from July 2012 to Jun 2013. His research areas include information security, network security, wireless mesh/sensor networks, and the future Internet.