

## RESEARCH ARTICLE

# New Network Security Method in Turing-Complete Cryptocurrency: User Accountability System

Sungil Kim<sup>\*†</sup>

**Abstract.** Ethereum was the first to apply a Turing-complete language to smart contracts in a cryptocurrency. It also introduced the gas system to protect its network against infinite loop attacks, which were made possible by this language. However, this gas system has two fatal weaknesses: it is vulnerable to spam attacks and network usage is reduced due to its strong usage-based characteristics. This paper proposes a user accountability system as a new network security method to solve this problem. The key to this system is forcing users to impose accountability for network security. Specifically, they must pre-deposit and lock their contract deposit, to be responsible for their actions in this system, and they may then execute smart contracts on the network. This locked contract deposit strengthens network security, because when the cryptocurrency network is attacked, they may lose some or all of their contract deposit. This is the basic principle of improving network security. The user accountability system aims not only to prevent spam attacks, but also to lower contract fees, both of which are in conflict with the gas system.

## KEY WORDS

1. Spam attacks.
2. Network security method.
3. Gas system.
4. Contract deposit method.
5. Ethereum.

## 1. Introduction

Bitcoin as a cryptocurrency is a replication system in which the distributed nodes constitute a peer-to-peer (p2p) network for online payment without relying on a trusted central authority.<sup>1</sup> It keeps a public and distributed blockchain up to date using a proof-of-work algorithm<sup>1</sup> and makes online payments using a *turing-incomplete language* that does not include loop operations, etc.<sup>2, 3</sup> Turing-incomplete languages typically do not include loop or branch operations or storage; if these are included, it can be generally called a *Turing-complete language*.<sup>3</sup> Note that the Turing-incomplete language on Bitcoin was purposefully adopted to protect its network from *infinite loop attacks*.<sup>2</sup> However, it suffers from inherent weaknesses that prevent users from coding complex logic.

---

<sup>†</sup>Sungil Kim, Ph.D. (hanaloum@gmail.com) is a researcher studying a cryptocurrency, Korea.

\*14QUhtKW329SqbfkNdNhtZgpWUoUorQdV

Ethereum, which inherited the concept of the distributed blockchain, seeks to overcome this by intentionally adopting a Turing-complete language.<sup>4</sup> Thus, the nodes that make up a p2p network of Ethereum can execute a *smart contract* through the Ethereum Virtual Machine (EVM). Here the smart contract (or *contract*) is the set of computer protocols that execute the terms of a contract.<sup>12</sup> In other words, it is any collection of bytecodes that have the ability to read/write to the blockchain and execute commands exactly as they are coded.<sup>13</sup> It can also read other contracts, make decisions, send internal tokens and execute other contracts.<sup>13</sup> Therefore, users can code smart contracts using the Turing-complete language that Ethereum provides and execute them by interacting with the blockchain. Therefore Ethereum provides a distributed platform that allows users to execute various smart contracts.<sup>4</sup> Additionally, the accounts in Ethereum have internal storage that records the contract's bytecodes and data.<sup>8</sup> The Turing-complete cryptocurrency this paper refers to is Ethereum.

In Section 2, we examine the gas system, which is Ethereum's network security method, and its problems. In Section 3, we propose a new user accountability system and a new contract deposit method to implement this system. We discuss the advantages of a user accountability system in Section 4 and Section 5 concludes this paper.

## 2. Ethereum's Security Method: The Gas System

The Ethereum network can become unstable because of *infinite loop attacks* conceived from the Turing-complete language. If attackers can use infinite loops in a network with little cost, they can rapidly increase the execution of their contracts, which can considerably slow the network or at worst stop it. Thus, Ethereum has introduced *gas*, which is a fee that is consumed at each execution step of a contract, the purpose of which is to protect the network from malicious attacks or abuse.<sup>5,9</sup> We call this fee a *contract fee*. Thus, the gas of Ethereum is the contract fee. This is different from the *transaction fee* paid for online payments. Note that gas is a key feature that protects the Ethereum network from malicious attacks and abuse. We call this a *gas system*, or *usage-based charging system*, as it protects the network with usage-based charges.

Specifically, the gas of Ethereum is designed to be consumed at each execution of a contract while being executed by the EVM. Each bytecode of the contract consumes a predetermined amount of gas at each execution step<sup>9</sup> and each bytecode has a different gas price depending on the network load or similar. For example, when an addition (ADD) operation is executed, it consumes three gas. The gas of the gas system is referred to as *fuel* because it is charged in proportion to the resource usage when a contract is executed.

### 2.1 Problems with the Gas System

Cryptocurrencies such as Bitcoin and Ethereum are characterized by pseudonymity because users can be only distinguished by their private keys and can also immediately create new ones without restriction. This pseudonymity has a nice feature that makes users' transactions

private in the cryptocurrency. However, it is also well known that attacks on networks stem from this, because it frees attackers from accountability for network attacks. Therefore, Ethereum's gas system cannot fundamentally solve spam attacks because of the pseudonymous nature of the cryptocurrency.

Actually, there were serious spam attacks on the Ethereum network in November 2016 that made the rate of block creation about 2–3 times slower, so they had a large impact on the network.<sup>6</sup> Ethereum developers implemented two hard forks to solve these attacks: 1) They raised the gas price of the operations used in the spam attacks and 2) They removed the empty accounts that were used.<sup>7</sup> Here, the hard fork was a change to an Ethereum protocol that made previously invalid blocks/transactions valid, and so there was a permanent divergence from the blockchains that resulted from non-upgraded nodes.<sup>14</sup>

However, their response was temporary because there is currently no obvious way to defend against these spam attacks besides raising the corresponding gas price. In addition, this approach will likely reduce network utilization because of the side effect of raising the contract fee. Thus the gas system has shown limitations in its capacity to protect the network from spam attacks. In fact, these attacks led to much discussion of how to adjust the gas price of the operations that were used in the attacks.<sup>10</sup>

Another disadvantage of the gas system is that it has the *strong usage-based characteristics* in which cost increases in proportion to network usage. It is important to know that the possibility of spam attacks and low gas prices is in conflict in the gas system because of these characteristics. These characteristics mean that users are likely to be afraid that the code length of their contracts is too long. This can occur if their contracts have many commands and will thus require a lot of money to execute. Ethereum tries to provide a platform on which users can easily execute various contracts. However, strong usage-based characteristics can prevent Ethereum from acting as a platform. A platform is more activated when many users are able to use it at little or no cost. This may be a problem in proof-of-stake (PoS) algorithms where contracts can make heavy use of node's computing resources.

### 3. New Idea: User Accountability System

This paper proposes a new user accountability system (or our system) to solve the gas system's serious problems. The key to our system is to force the users to impose accountability for network security. If they follow accountability or the rules imposed by our system, the contract fee they must pay will be much less than that of the gas system, which will mean that they can obtain economic benefits when using the network at a small cost. Conversely, if they violate the rules, they will suffer economic damage. It is important that our system maintains network security with the economy principle. In other words, our system is a new network security method that forces users to suffer economic damage if they burden the network with malice or carelessness. Thus, we emphasize that the user accountability system not only prevents spam attacks, but also lowers the contract fee, both of which are in conflict

in the gas system. We will now introduce a new contract deposit method with which to implement it.

### 3.1 Contract Deposit Method

Our user accountability system is designed to force users to pre-deposit their *contract deposit* to be responsible for their actions. If the contract deposit is not pre-deposited, users cannot execute their contract on the network; we call this a *contract deposit method*. Specifically, they must pre-deposit their contract deposit in a special account before executing their contracts, and this is then locked so that the deposit may not be withdrawn. In a specific way, users can lock their contract deposit directly on their wallets. The contract deposit is intended to confirm that they will not attack the network; these two systems are completely different: Our system is based on the contract deposit method, whereas the gas system is based on the strong usage-based characteristics.

In the real world, in most cases in which the actors break the rules, they suffer damage in some form, which is its most unique feature. Likewise, if users break the rules of our system, they suffer economic losses. This is the basic principle of improving network security; it shows that this contract deposit method alleviates the pseudonymity of cryptocurrency. Specifically, it allows the network to be strongly protected while also allowing users to obtain economic benefits from cheap contract fees during the execution of their contracts.

In addition, the method of maintaining security with the user's deposit is already used in a Proof-of-Stake (PoS) algorithm such as tendermint, etc.,<sup>11</sup> which we call a *PoS deposit*. However, the purpose of each is completely different. The contract deposit proposed in this paper is used to protect the cryptocurrency network from spam attacks, etc. whereas the PoS deposit is only used to secure the consensus algorithm.

We will now introduce two specific contract deposit methods: 1) A gas-based method that uses gas limits as the rules, and 2) a post-verification method. We also want to say that our system may be implemented in many other ways that are not mentioned in this paper.

#### 3.1.1 Gas-based Method

Fig. 1 shows examples of the gas system of (a) Ethereum and (b) our system while the contract is executed. It is assumed that the gas in the figure is constantly used (or consumed) over time in all blocks. Of course, the figure does not represent the actual situation, because the gas prices differ for each operation and contracts are not normally executed equally in all blocks.

Let's us first examine the gas system of Ethereum (Fig. 1a). In the gas system, gas must have strong usage-based characteristics because it is entirely responsible for network security. In other words, the gas system protects its network by consuming more fees for the greater executions of a contract. The solid line in Fig. 1a is gas, which represents the total contract fee consumed (paid) during a contract's execution. In addition, users in the gas system must first

set their gas limit, which is the maximum amount of gas that can be consumed, and then execute a contract. If their gas limit is exceeded contract's execution, all executions are reverted (rolled back) and the consumed gas is not returned.

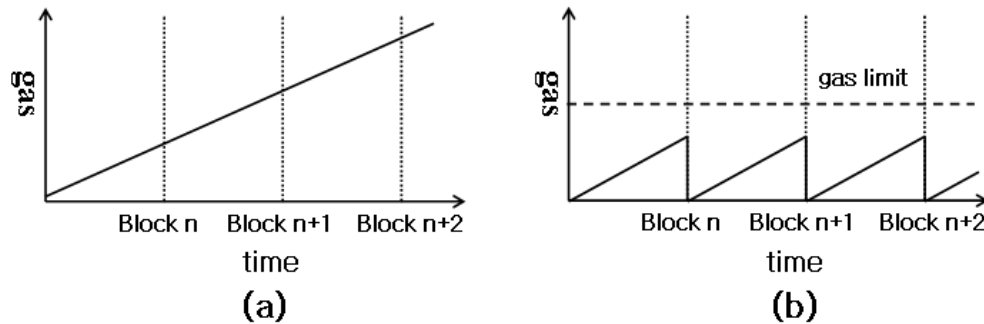


Fig. 1. (a) The gas system of Ethereum: the solid line is gas, which represents the contract fee paid by users. (b) User accountability system (our system): the solid line is gas, which only indicates whether the rules of our system have been followed, and the rules do not exceed the gas limit in the figure. In addition, the contract fee (not shown) is paid once when the contract is first executed for each block.

Now we will examine a contract deposit method for network security. We first propose a *gas-based method* that is based on the gas system. Fig. 1b is an example of this method, and shows how it works. In the figure, gas is initialized and accumulated for each block. The solid line in Fig. 1b is gas, which only serves to indicate whether the rules of our system have been followed. In other words, the gas in Fig. 1b is used only as an indicator of compliance with the rules of our system. Therefore, the role of gas of our system is completely different from that of the gas system. Although the contract fee is not shown in this figure, it is paid once when a contract is first executed for each block. If the peak of the gas does not exceed the gas limit, users are charged a very low contract fee for each block compared to in the gas system. Therefore, *the rules of our system (Fig. 1b) do not exceed the gas limit*. The gas limit in the figure is automatically calculated according to the size of the user's contract deposit; the greater the contract deposit, the higher the gas limit. In addition, our system continuously compares the gas limit and the gas peak for each block during the execution of contracts. Furthermore, the minimum contract deposit in our system must be predetermined. The minimum contract deposit refers to the smallest deposit that is locked before using the network. Thus, this minimum deposit should be determined by comprehensively considering the various issues that are involved.

In the example (Fig. 1b), gas is initialized for each block, but it can also be initialized and accumulated for each group of blocks. Thus, the contract fee in our system can be charged once for each block or each group of blocks. It is also charged only once, unlike the gas system, when there is a first run on each block or each group of blocks.

This gas-based method can enhance network security in other ways such as by applying progressive penalty methods to our system. Specifically, if attackers continue to attack the network, their contract deposit will be forfeited more quickly due to this progressive nature. Therefore, users need a way in which their contracts do not exceed the gas limit. Solving this problem requires providing them with corresponding commands or functions that can be used in their contract codes. Here are some examples; first, they can code their contract to temporarily stop upon arriving at their gas limit and then execute the remainder after their gas has been initialized, as shown in Fig. 1b. Another way involves coding their contract to automatically deposit additional deposits upon arrival, and then enabling the remainder to be executed.

In addition, users in our system will likely exceed their gas limit. In this case, a certain percentage of their contract deposit should be forfeited as a penalty. However, if this is due to carelessness, this problem can be solved by how they deposit their additional deposits. Here, the additional deposit refers to both the contract deposit that corresponds to the excess of their gas limit and a fine deposit that is added as a fine. In this case, if they deposit their additional deposit within a specified time after violating the rules, our system can give them the opportunity to be exempted from the penalty and continue benefiting from the low contract fee. The fine deposit here has the same purpose as the contract deposit, but it is an additional locked deposit due to their mistake. However, if users attack the cryptocurrency network, they are not given this benefit and a certain percentage of their contract deposit will be taken as a penalty.

Spam attacks are characterized by attacking the network using some commands. Actually, the `EXTCODESIZE` command was used for spam attacks in November 2016.<sup>6</sup> Therefore, measuring the load on the network and varying the gas price of commands that used in attacks is possible because of these characteristics. This can be easily implemented in the gas-based method. In addition, the gas-based method can be easily implemented by using the gas system employed by Ethereum, and it is a greatly helpful for calculating gas correctly when compiling contracts. Ethereum developers are developing the viper programming language, which is similar in syntax to Python, which allow precise gas computations at the compile stage.<sup>15</sup> In addition note that the contract deposit in our system (Fig. 1b) is not consumed, and the contract fee is consumed during the contract's execution. Unlike this, the contract deposit can be designed so that a predetermined portion of it is consumed as a contract fee.

When the PoS algorithm is applied to Ethereum in the future, the nodes will rarely use their computational resources to maintain the network, unlike in the current proof-of-work (PoW) algorithm. In this case, users can run contracts that use many resources, and the gas based method can then easily be applied.

### 3.1.2 Post-Verification Method

Second, our system can be implemented via a *post-verification method* that is also based on

contract deposits. In this method, if attackers maliciously attack the network, participants with certain conditions validate these attacks afterwards and then forcefully forfeit some or all of their contract deposit according to the predetermined voting procedures. For example, the penalty in our system can be decided by the votes of participants. Here, participants may represent a verification group or developers or others involved in the cryptocurrency.

In this method, *the rules of our system do not attack the network*. Of course, it would be nice to automatically detect network attacks in real time, but it would be very difficult to implement. Therefore, it is currently a good idea to find attackers after their network attacks and then impose economic damage on them. In the case of malicious network attacks, a large portion of their contract deposit should be forfeited so that they feel a great fear. This is because this method aims to make attackers predict their losses, and thus makes them very afraid of attacking the network. Therefore, the more damage attackers do to the network, the greater the percentage of their contract deposit they forfeit. It is also important that the penalties for network attacks are predetermined and publicly notified so that attackers are clearly aware of them. In addition, it is good idea to use both the gas system and the post-verification method together.

In the contract deposit method, the withdrawal of the locked deposit should be designed to take, for example, longer than three weeks or two months for the penalty of our system.

#### 4. Advantages of the User Accountability System

Our user accountability system has significant strengths because it maintains the network's security based on economic principles. The following outlines our system's strengths compared to Ethereum's gas system.

*Reduce attackers' malicious attacks*—Attackers try to attack the weaknesses of the network in Turing-complete cryptocurrencies. However, users who intentionally attack the network suffer economic damages. Therefore, malicious attacks on the network are naturally reduced significantly. Our system can greatly enhance network security via economic principles.

*Strengthen the platform's role by increasing network usage*—The users in our system can use the network with cheaper contract fees than the gas system. Therefore, our system significantly reduces the pressure on them when executing contracts that have long code lengths. This can greatly increase the overall network usage in the PoS algorithm; the greater the network's total usage, the more the platform's role is strengthened.

*Stabilize the internal token price*—One factor supporting the internal token price in a cryptocurrency is the token's usability. As the usage of the token increases, its price can be more stable. The users in our system must lock their contract deposit to execute their contract on the network. Thus, the use of internal tokens can be greatly increased, which makes it possible to stabilize the price of tokens.

## 5. Conclusion

This paper proposes a user accountability system as a new network security method in Turing-complete cryptocurrencies. The main feature of our system is that the user is responsible for the network security. Contrary to this, Ethereum's gas system is based on strong usage-based features that protect the network by charging fees at each stage of a contract. Meanwhile, our system forces users to lock a contract deposit and then executes their smart contracts on the network. If they break the rules, some of their contract deposit is forcibly forfeited. Thus, our system provides a way to reduce attackers' willingness to attack the network via economic principles; this not only prevents spam attacks, but also lowers the contract fee, both of which are in conflict in the gas system.

We also propose the contract deposit method to implement our system. We have presented two concrete methods of implementing this method: 1) a gas-based method that uses the gas limit in its rules and 2) a post-verification method. When the PoS algorithm is applied to Ethereum in the future, users can run contracts that have high computing resource requirements, and the gas-based method can be easily applied.

Now let's think Sybil attacks. Cryptocurrency is a pseudonymous system, so it is fundamentally vulnerable to Sybil attacks. Our system has very strong characteristics for dealing with serious spam attacks. However, it is also vulnerable to Sybil attacks. We think that preventing Sybil attacks may require adjusting the contract fee depending on the average load on the cryptocurrency network, as discussed in Ethereum. This contract fee can be adjusted at a constant rate for all commands or each bytecode being attacked.

## Notes and References

<sup>1</sup> Satoshi Nakamoto. "Bitcoin: A Peer-to-Peer Electronic Cash System." *No Publisher* (2009)

<https://bitcoin.org/bitcoin.pdf>

<sup>2</sup> No Author. "Script." *Wikipedia* (accessed 5 April 2017) <https://en.bitcoin.it/wiki/Script>

<sup>3</sup> No Author. "Turing-completeness." *Wikipedia* (accessed 5 April 2017) [https://en.wikipedia.org/wiki/Turing\\_completeness](https://en.wikipedia.org/wiki/Turing_completeness)

<sup>4</sup> No Author. "Ethereum." *Wikipedia* (accessed 5 April 2017) <https://en.wikipedia.org/wiki/Ethereum>

<sup>5</sup> Gavin Wood. "Ethereum: A secure decentralized generalized transaction ledger" *No Publisher* (2014) <http://gavwood.com/paper.pdf>

<sup>6</sup> Vitalik Buterin. "Transaction spam attack: Next Steps." *Ethereum blog* (2016) <https://blog.ethereum.org/2016/09/22/transaction-spam-attack-next-steps/>

<sup>7</sup> Martin Swende. "Announcement of imminent hard fork for EIP150 gas cost changes." *Ethereum blog* (2016) <https://blog.ethereum.org/2016/10/13/announcement-imminent-hard-fork-eip150-gas-cost-changes/>

<sup>8</sup> No Author. "Ethereum white paper." *Github* (accessed 5 April 2017) <https://github.com/ethereum/wiki/wiki/White-Paper>

- <sup>9</sup> Ethereum. “Gas Economics.” *Github* (accessed 5 April 2017) <https://github.com/LeastAuthority/ethereum-analyses/blob/master/GasEcon.md>
- <sup>10</sup> Vbuterin. “On Gas Price Markets.” *r/Ethereum* (accessed 5 April 2017) [https://www.reddit.com/r/ethereum/comments/572n4q/on\\_gas\\_price\\_markets/](https://www.reddit.com/r/ethereum/comments/572n4q/on_gas_price_markets/)
- <sup>11</sup> Jae Kwon. “Tendermint: Consensus without Mining.” *Tendermint docs* (2014) <https://tendermint.com/static/docs/tendermint.pdf>
- <sup>12</sup> No Author. “Smart contract.” *Wikipedia* (accessed 5 April 2017) [https://en.wikipedia.org/wiki/Smart\\_contract](https://en.wikipedia.org/wiki/Smart_contract)
- <sup>13</sup> No Author. “Contract tutorial.” *Github* (accessed 5 April 2017) <https://github.com/ethereum/go-ethereum/wiki/Contract-Tutorial>
- <sup>14</sup> No Author. “Hard fork.” *Bitcoin wiki* (accessed 5 April 2017) <https://en.bitcoin.it/wiki/Hardfork>
- <sup>15</sup> No Author. “Viper.” *Github* (accessed 5 April 2017) <https://github.com/ethereum/viper>