Available online at www.ijpe-online.com

International Journal of Performability Engineering

vol. 15, no. 11, November 2019, pp. 3016-3023
DOI: 10.23940/ijpe.19.11.p21.30163023

Smart Contract Receipt based on Virtual Iterative Function

Yifeng Yin[a], Tingjun Zhang[a,*], Chaofei Hu[a], and Yong Gan[a,b]

[a]*School of Computer and Communication Engineering, Zhengzhou University of Light Industry, Zhengzhou, 450002, China*
[b]*Zhengzhou Institute of Technology, Zhengzhou, 450044, China*

Abstract

Smart contracts are the most important feature in block chain applications, and they are also the main reason why blockchains are called disruptive technology. Traditional intelligent contracts with receipts are generated by SHA-256A UXTO (unexpended transaction output), and increasing the number of receipts slows down the speed. This paper introduces the operation of receipts in smart contracts and proposes to generate contract receipts with the VIF virtual iteration function. VIF takes advantage of the excellent features of the Hash function and the unreadable nature of the self-compiled system, so that different contract parameters generate unique and non-repudiation receipts through the virtual iterative function, providing a secure and reliable credential for smart contracts. Finally, the speeds at which the VIF receipt and traditional UXTO receipt are generated are compared.

Keywords: smart contract; virtual iterative function; receipt; non-repudiation; UXTO

(Submitted on September 15, 2019; Revised on October 12, 2019; Accepted on November 25, 2019)

1. Introduction

Blockchain technology (also known as distributed ledger technology) has been the focus of attention at home and abroad in recent years. A smart contract is a computer program that automates the execution of contract terms. It has been creatively proposed that "a smart contract is a computable transaction agreement that enforces contract terms" [1].

There are many informal definitions of smart contracts, and Szabo stated that "smart contracts use contract and user interfaces to facilitate contract execution" [2]. Miller believed that smart contracts are contracts written in program code, and its terms are enforced by the program [3-4]. Ethereum's smart contracts are blockchain-based programs that directly control digital assets [5]. In general, a smart contract is a set of commitments defined in digital form, including agreements on which contract participants can execute these commitments. It is a computer program that automatically enforces contract terms and is deployed on shared, replicated books. It maintains its state, controls its assets, and responds to incoming external information or assets. Commitment refers to the rights and obligations agreed by the contract participants, which define the nature and purpose of the contract. The digital form means that the contract has to be written in computer readable code [6-7]. As long as the parties reach an agreement, the rights and obligations established by the smart contract are performed by a computer or computer network. The agreement is a technical realization, on the basis of which the contractual commitment is fulfilled, or the contractual commitment to the realization is recorded [8-9].

The DAO smart contract, which was run on the Ethereum public chain in 2016, was subjected to a serious redirection attack, and the public funds raised by the contract were continuously redirected to a subcontract, which involved a total of more than three million Ethereum [10]. DAO is essentially a risk investment fund, which was raised through Ethereum and will be locked into smart contracts. No one can use the money alone. The event was triggered by the exploitation of the scripting vulnerability of the DAO smart contract itself. Therefore, the security and credibility of smart contracts have gained widespread attention. Improving the security and reliability of smart contracts has become a problem that still needs to be solved [11-12].

The traditional smart contract receipt involves consuming several UXTOs created by the previous transaction and then

* Corresponding author.
E-mail address: zhangtingjun@163.com

generating one or more new UTXOs. These newly generated UTXOs can also be consumed by future exchanges. As the number of receipts increases, the generation speed of UTXOs becomes slower [13].

VIF (virtual iterative function) technology was introduced to quickly generate high security, high reliability, and non-repudiation [14-16]. VIF takes advantage of the excellent features of the Hash function and the unreadable nature of the self-compiled system, so that different contract parameters generate unique and non-repudiation receipts through the virtual iterative function [17-19].

2. Receipt Generation Model

2.1. Virtual Iterative Function

The function set F is constructed with a plurality of one-way functions, which internally contain n candidate single term functions. Take the virtual iterative function VIF ( ) as the formal parameter. The user participating in the contract, Alice, initially holds her own control key, the array parameter from the contract parameter hash, and the array b argument obtained by the user Bob. When the user does not substitute the initial key to the function set F, VIF ( ) does not have an iterative function. When the complete argument is substituted into the function set, according to the object-oriented overload theory, the function is the argument control function. The single-item functions of the n candidate candidates constitute a virtual iterative function shared by both parties with equal selected probabilities.

The symbols used in this article are shown in Table 1.

Table 1. Main symbols and their meanings

Sign	Meaning
k_a, k_b	Users a and b initially have a secure control key
F	Self-compiler constructed from multiple candidate one-way functions
p_a, p_b	Generate your own random parameter sets using their respective contract parameters
$H(\ )$	One-way hash function mapping
I_a, I_b	Half of the iterative function
$VIF(\ )$	Complete virtual iterative function after negotiation

The virtual iterative function preconditions are constructed as follows:

- Suppose the two sides of the contract are Alice and Bob, respectively. They each have a secure control key k_a and k_b, which are used to call the permutation function to construct a secure sub-algorithm set.
- Alice and Bob have the same candidate one-way function constructor from the compiler F.
- Alice and Bob use their respective contract parameters to generate their own random parameter sets p_a and p_b.

The steps to construct a safe virtual iterative function are shown in Figure 1.

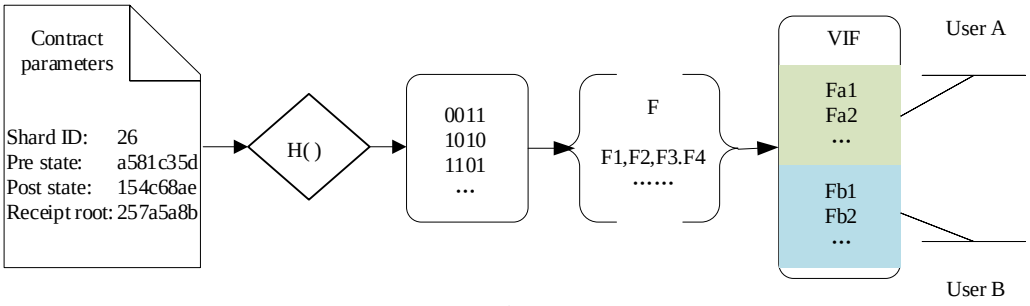

Figure 1. Virtual iterative function running diagram

The steps to construct a safe virtual iterative function are as follows:

- Both parties use the same self-compiler to combine the random parameter sets p_a and p_b generated by the respective contract parameters to obtain the respective output arrays.

- Alice will obtain the output array with the function set F and obtain the upper half iteration function (1).

$$I_a = \left(f_a, F_y \right) \tag{1}$$

- Similarly, Bob uses his own output array to obtain the lower half iteration function (2).

$$I_b = \left(F_x, f_b \right) \tag{2}$$

- The two sides use the public channel to exchange the half iteration functions (1) and (2) generated by each.

- Both parties bring their own output array into the received half iteration functions (1) and (2) and obtain a complete virtual iterative function (3) for generating contract receipts.

$$VIF\left(a,b\right) = \left(f_a, f_b \right) \tag{3}$$

2.2. Smart Contract Receipt Mechanism

The implementation of the transaction by the smart contract can change the state of the partial slice, but it will also generate a "receipt", which is stored in a shared storage space and can be viewed later through the transaction execution of other fragments (but cannot be modified).

The contract receipt mechanism is divided into several stages, as shown in Figure 2.

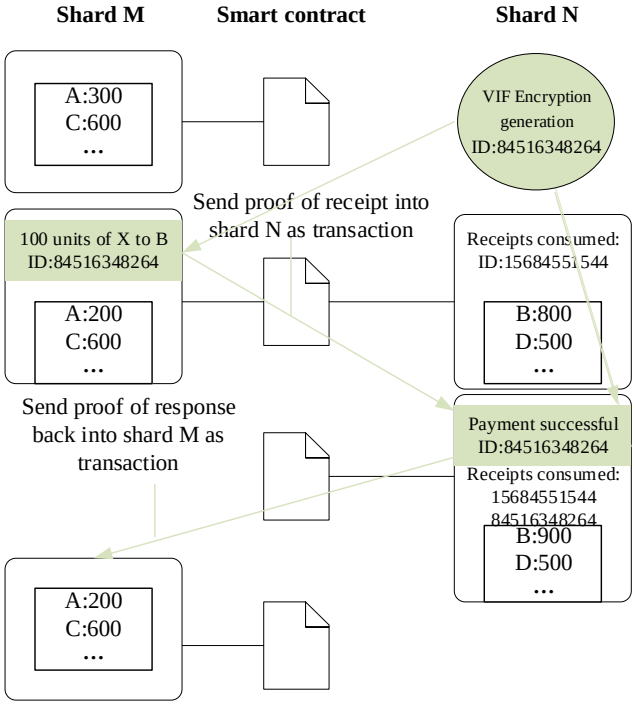

Figure 2. Contract receipt mechanism

In shard M, the balance of A is reduced by 100, and a receipt is generated by the VIF virtual iteration function that says that "B should receive 100 units of X tokens. The ID number of this receipt is 84516348264."

In shard N, the evidence that the receipt is created is introduced in the form of a transaction (the example in the private chain will be a pointer to the shared storage module, and in the case of the public chain it will be evidence). This contract will verify the following details: (i) the receipt is correct, and (ii) the receipt of the same ID number has not been consumed. This contract adds a balance of 100 to B and marks the store with the fact that the receipt and its ID number have been consumed.

In more advanced applications, if shard M is running a smart contract, it will perform some operations after the payment is completed. Thus, the record in shard N can be used as a receipt, provided that the payment is completed, so that shard M can continue the operation.

3. Smart Contract Receipt

3.1. Receipt Encryption

To achieve communication between slices, receipts are a key messaging mechanism. A receipt is a voucher for the execution of other shards. Especially important are its security, certainty, and non-repudiation. We use the virtual iterative function to generate the contract receipt, which requires the contract parties to share a function set constructed by multiple one-way functions, with the initialization of the virtual iterative function as the formal parameter and the array mapped by the contract user control key and the contract parameter hash as the actual parameter. The user will not bring the arguments into the function set, and the virtual iteration function does not have the function of generating the receipt. When both users bring their actual arguments into the function set, the function of the argument control function, together with the virtual iterative function, can generate the contract receipt. The generation of contract receipts can be divided into six steps, as shown in Figure 3.

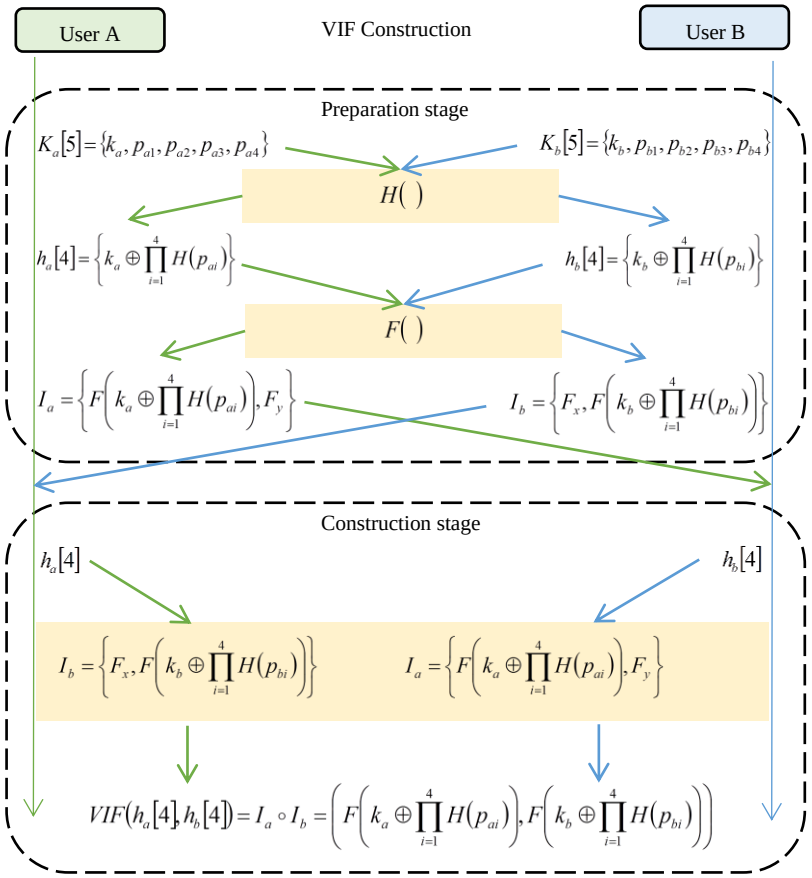

Figure 3. Virtual iterative function construction process

Step 1 User A and User B, who need to establish a session, use the randomness of the contract parameters to fetch four random numbers together with their own private key to form a five-element array as the new private key groups (4) and (5).

$$K_a[5]=\{k_a,p_{a1},p_{a2},p_{a3},p_{a4}\} \tag{4}$$

$$K_b[5]=\{k_b,p_{b1},p_{b2},p_{b3},p_{b4}\} \tag{5}$$

Step 2 User A brings the obtained quintuple private key into the one-way function $H(\ )$ to obtain a partial parameter (6).

The self-compiler F that brings in the candidate one-way function constructs shared by both parties obtains the upper half-iteration function (7). Similarly, User B brings the generated quintuple private key into the one-way function $H(\ )$ containing the self-compilation system to obtain another part of the parameter (8) and brings the self-compiler F constructed by the candidate one-way function shared by both sides to obtain the lower-half iterative function (9).

$$h_a[4]=\left\{k_a \oplus \prod_{i=1}^{4} H\left(p_{ai}\right)\right\} \tag{6}$$

$$I_a=\left\{F\left(k_a \oplus \prod_{i=1}^{4} H\left(p_{ai}\right)\right), F_y\right\} \tag{7}$$

$$h_b[4]=\left\{k_b \oplus \prod_{i=1}^{4} H\left(p_{bi}\right)\right\} \tag{8}$$

$$I_b=\left\{F_x, F\left(k_b \oplus \prod_{i=1}^{4} H\left(p_{bi}\right)\right)\right\} \tag{9}$$

Step 3 User A and User B use the public channel to send each of the generated half iteration functions I_a and I_b to the other party.

Step 4 After receiving the lower-half virtual iterative function I_b sent by User B, User A brings his or her own parameter $h_a[4]$ into the lower-half virtual iterative function to obtain a complete virtual iterative function (10). Similarly, after receiving the lower-half virtual iterative function I_a sent by User A, User B brings his or her own parameter $h_b[4]$ into the upper-half virtual iterative function to obtain a complete virtual iterative function VIF. Then, the virtual iterative function negotiation between the two parties is completed.

$$VIF\left(h_a[4], h_b[4]\right)=I_a \circ I_b=\left(F\left(k_a \oplus \prod_{i=1}^{4} H\left(p_{ai}\right)\right), F\left(k_b \oplus \prod_{i=1}^{4} H\left(p_{bi}\right)\right)\right) \tag{10}$$

Step 5 User A and User B use the negotiated virtual iteration function VIF to generate a long-term encryption sequence for the contract as the receipt of the contract.

Step 6 The contract verifies the receipt: (a) the receipt is correct, and (b) the receipt with the same ID number has not been consumed. After passing the verification, the contract will continue to execute and mark the fact that the receipt and its ID number have been consumed in the storage area. After the contract is completed, User A and User B automatically destroy the previously generated random parameter group and private key group, ready to generate the receipt of the next round of contracts.

3.2. Encryption Extension

It can be seen from Section 3.1 that the basis of the receipt encryption is that User A and User B each grab four random numbers, together with their own private keys, to form two columns of five-element arrays. Therefore, before the completion of the virtual iterative function negotiation, there are 25 (5×5) different choices for the composition of multiple single-function functions in the self-compiler.

In order to improve the security of the virtual iterative function, we extend the random seed number of Users A and B and can increase the random seed according to the security requirements, as shown in Figure 4.

If the random seeds of Users A and B are increased to "n", there are $n\times n$ constituent methods when the two parties negotiate to complete the virtual iterative function. User A's random seed is used as the Y axis, and User B's random seed is taken as the X axis. Then, the VIF negotiation result can be any integer point on the two-dimensional coordinate graph. When a contract is completed, both users automatically destroy the previously generated random seed and encryption functions and regenerate them. The encryption function of the next receipt is $n\times n$ possibilities once again.

The increase in the number of random seeds increases the compositional variation of the iterations in the case where the single function *H*( ) contained in the compiler is limited. The number of types of iterations is directly proportional to security.

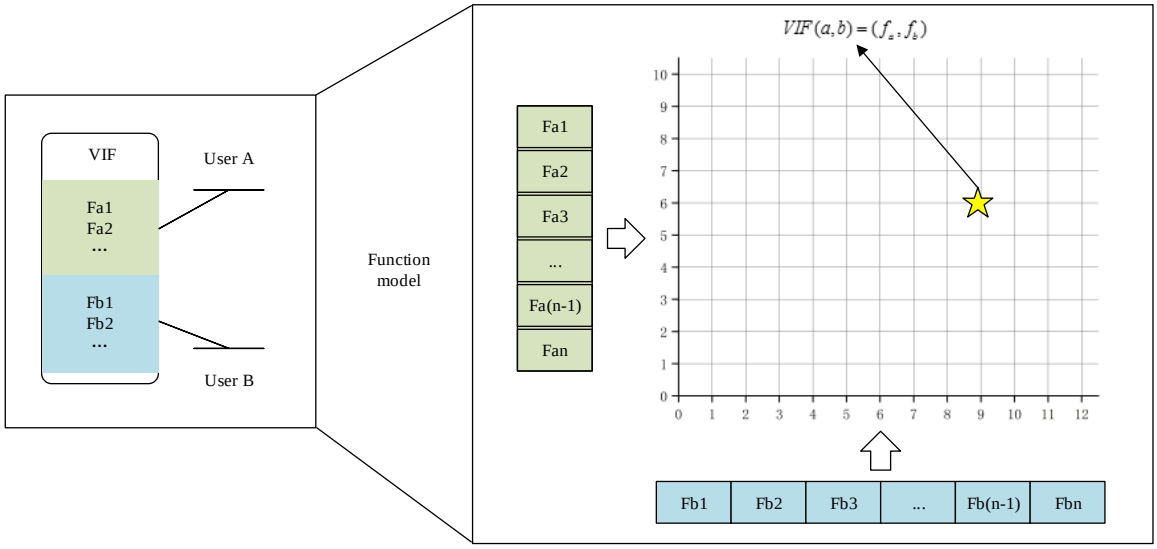

Figure 4. Virtual iterative function model

4. Experiment

We used Python 3.7 to simulate traditional receipt generation and VIF receipt generation on a personal computer. The experimental environment is described as follows:

- CPU: AMD Ryzen 5 2600X Six-Core Processor 3.6GHz
- RAM: 8.00GB
- OS: Windows 10

4.1. Experimental Comparison

In the traditional smart contract receipt generation process, a hash algorithm called SHA256 is used. SHA-256 is an improved version of the SHA-1 algorithm. The maximum length of the input message of the SHA-256 algorithm does not exceed 2^64 bits. The input is processed by a 512-bit packet, and the output is a 256-bit packet digest. Because the traditional receipt is UXTOs, in order to ensure that the resulting receipt meets the requirements, a limit is imposed on the generated message. The qualification conditions are checked first, and then the check is repeated. If they are the same, the input message is regenerated and hashed again. If they are different, the receipt is generated successfully and stored.

VIF-based receipt generation builds a single function set F. SHA-1 is not secure in terms of hash mapping, and the SHA-256 algorithm as an improved version of SHA-1 is temporarily safe, so the hash mapping also uses SHA-256. The input message is a different simulation contract parameter. First, a virtual iterative function is constructed, and then the message is input into a receipt. Similarly, if the receipt is in conflict, the conflict is recalculated, and if it is not, the receipt is stored.

In order to fully simulate the UXTOs generation process, we give a limited definition of the message digest generated by SHA-256. A qualified message digest must be checked to further assess reproducibility. The experimental results are compared as shown in Figure 5. Because SHA-256 is fast to generate, the speed is high when there are few receipts. However, because UXTOs need to limit one condition, the more required receipts there are, the more conflicts there are. The more operations that are performed, the longer the receipt will be. When a certain number is reached, that is, when the number of keys is between 400 and 500, the VIF receipt generation rate without a limit exceeds the traditional receipt generation speed. Although the number of candidate one-way functions from the compiler F is large, since these candidate one-way functions adopt the polynomial cryptography theory of the stream cipher mechanism, the time complexity of the VIF virtual iterative function generation is small.

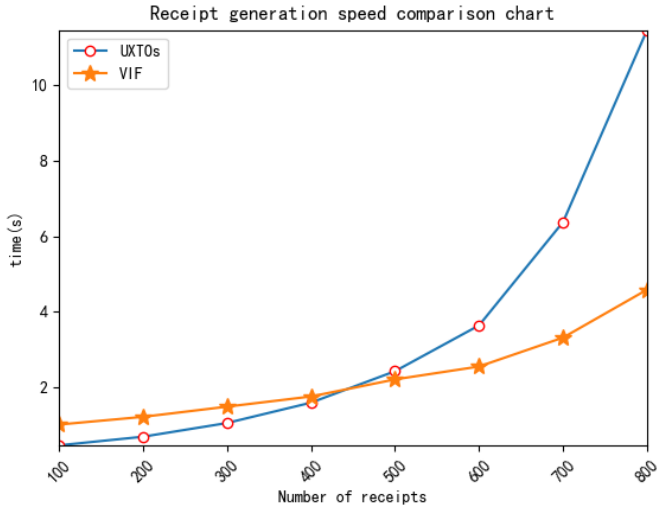

Figure 5. Receipt generation speed comparison chart

4.2. Security Analysis

Suppose a potential enemy attempts to intercept the encrypted function negotiated by both parties by eavesdropping on the information transmitted on the public channel (that is, by obtaining the private key of any party and the random seed) in order to decipher the VIF function generated by the contractual parties, thereby seeking to obtain the receipt of the entire contract. The system is generated. Since the random number obtained by the two parties from the network cannot be acquired again at a certain moment, even if the enemy acquires the keys of both parties of the communication, the random number seed that he or she needs cannot be guessed.

If one of the enemy's counterfeit contracts tries to implement a man-in-the-middle attack, it cannot be achieved. In any case, the attacker can only obtain half of the virtual iterative function, so it is impossible to obtain a virtual iterative function that generates contract receipts synchronized with both parties. In addition, the authentication between the contract parties depends on the user's stored and own contract conditions. As for the corresponding information, the enemy cannot obtain the contract key means that the contract information of the user sent after completion of the virtual iterative function negotiation is safe and cannot be taken by the enemy.

At the same time, using the mechanism of the virtual iterative function, both parties of the contract share the same iterative function compiling and generating device. Therefore, both parties form a contract receipt, which has non-repudiation characteristics.

5. Conclusions

In this article, we have addressed a shortcoming of traditional UXTOs receipts. The virtual iterative function is used to generate the contract receipt, and the excellent characteristics of the Hash function and the unreadable characteristics of the self-compiled system are used to make different contract parameters generate the only non-repudiation receipt through the virtual iterative function. A simulation experiment of the receipt generation is carried out. Compared with a larger number of receipts, the VIF receipt generation has the advantage of speed.

Acknowledgements

This work is supported by National Natural Science Foundation of China (No. 61572445, U1804263, and 61272038).

References

1. M. Pilkington, “11 Blockchain Technology: Principles and Applications,” *Research Handbook on Digital Transformations*, Vol. 225, 2016
2. I. Grishchenko, M. Maffei, and C. Schneidewind, “Foundations and Tools for the Static Analysis of Ethereum Smart Contracts,” in *Proceedings of International Conference on Computer Aided Verification*, pp. 51-78, Springer, Cham, July 2018
3. A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts,” in *Proceedings of 2016 IEEE Symposium on Security and Privacy* (*SP*), pp. 839-858,

2016
4. E. Mik, "Smart Contracts: Terminology, Technical Limitations and Real World Complexity," *Law, Innovation and Technology*, Vol. 9, No. 2, pp. 269-300, 2017
5. K. Bhargavan, A. Delignat-Lavaud, and C. Fournet, "Formal Verification of Smart Contracts: Short Paper," in *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pp. 91-96, ACM, 2016
6. M. Corrales, M. Fenwick, and H. Haapio, "Legal Tech, Smart Contracts and Blockchain," Springer, 2019
7. P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Security: Practical Security Analysis of Smart Contracts," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 67-82, ACM, October 2018
8. D. Magazzeni, P. McBurney, and W. Nash, "Validation and Verification of Smart Contracts: A Research Agenda," *IEEE Computer*, Vol. 50, No. 9, pp. 50-57, 2017
9. H. Zhao, P. Bai, and Y. Peng, "Efficient Key Management Scheme for Health Blockchain," *CAAI Transactions on Intelligence Technology*, Vol. 3, No. 2, pp. 114-118, 2018
10. N. Atzei, M. Bartoletti, and T. Cimoli, "A Survey of Attacks on Ethereum Smart Contracts (sok)," in *Proceedings of International Conference on Principles of Security and Trust*, pp. 164-186, Springer, Berlin, Heidelberg, April 2017
11. K. Delmolino, M. Arnett, and A. Kosba, "Step by Step Towards Creating a Safe Smart Contract: Lessons and Insights from a Cryptocurrency Lab," in *Proceedings of International Conference on Financial Cryptography and Data Security*, pp. 79-94, 2016
12. Y. Chen, Z. Zhang, and B. Yang, "Research and Application of Warehouse Receipt Transaction based on Smart Contract on the Blockchain," in *Proceedings of 2018 International Conference on Mechanical, Electronic, Control and Automation Engineering* (*MECAE 2018*), Atlantis Press, March 2018
13. A. W. Appel, "Verification of a Cryptographic Primitive: SHA-256," *ACM Transactions on Programming Languages and Systems*, Vol. 37, No. 2, pp. 1-7, 2015
14. L. He, Y. Gan, and Y. Yin, "Secure Group Ownership Transfer Protocol for Tags in RFID System," *International Journal of Security and its Applications*, Vol. 8, No. 3, pp. 21-30, 2014
15. N. M. Smith, "Systems and Methods for Distributed Trust Computing and Key Management," *U.S. Patent*, No. 9, pp. 215-249, 2015
16. Y. Yin, Y. Yang, Y. Gan, and M. Zhang, "Researching Security Mechanisms of the Polymorphic Authentication Service Protocol," *Journal of Computational Information Systems*, Vol. 9, No. 7, pp. 2641-2647, 2013
17. Y. Yin, X. Li, and Y. Hu, "Fast S-Box Security Mechanism Research based on the Polymorphic Cipher," *Information Sciences*, Vol. 178, No. 6, pp. 1603-1610, 2008
18. Y. Yin, Y. Gan, H. Wen, and T. Li, "A Symmetric Key Exchange Protocol based on Virtual S-Box," *China Communications*, Vol. 11, No. 14, pp. 46-52, 2014
19. Y. Yin, R. Zhang, and Y. Gan, "Researching Intelligent Decision of Loop Tiling Scale based on PSO," *Journal of Computational Information Systems*, Vol. 9, No. 13, pp. 5449-5455, 2013