Hindawi
Security and Communication Networks
Volume 2023, Article ID 8455894, 12 pages
https://doi.org/10.1155/2023/8455894

Research Article

A Four-Tier Smart Contract Model with On-Chain Upgrade

Zhiqiang Du, Hao Cheng, Yanfang Fu, Muhong Huang, Liangxin Liu, and Yifan Ma

School of Computer Science and Engineering, Xi'an Technological University, Xi'an 710021, China

Correspondence should be addressed to Yanfang Fu; fuyanfang@xatu.edu.cn

Received 22 June 2022; Revised 2 December 2022; Accepted 26 December 2022; Published 30 January 2023

Academic Editor: Jie Cui

Copyright © 2023 Zhiqiang Du et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

When there are loopholes in smart contracts or changes in demand, the existing three-tier model can only implement partial on-chain upgrades and the security of on-chain upgrades cannot be guaranteed. In this study, we optimized the three-tier smart contract model and proposed a four-tier smart contract model that includes the proxy, verification, business, and storage layers. The proxy layer is used to link contracts with other layers, the verification layer is used to check the integrity, boundary values, and abnormal processes of contracts, the business layer is used to execute business logic, and the storage layer is used to store data uniformly. On the basis of the proposed model, an on-chain upgrade and verification algorithm is proposed, which implements on-chain upgrade, on-chain verification, and version compatibility of contracts. We then design an information exchange system based on the proposed model and algorithm and test it based on the FISCO BCOS platform. Experiments show that, compared to the three-tier model, the proposed four-tier model and algorithm can implement the on-chain upgrade and reduce the contract complexity and data migration cost at the cost of some overall deployment.

1. Introduction

Szabo first proposed the concept of smart contracts in 1994 and provided a classic definition, which is a computer program that can implement the terms of a contract [1]. It was difficult to ensure the correct execution of computer programs in a noncomplete trust environment before the emergence of blockchain, which led to smart contracts not getting much attention until the advent of Bitcoin in 2008 [2]. As the underlying technology of Bitcoin, the characteristics of decentralization, trustworthiness, and data immutability of the blockchain meet the requirements of smart contracts for operating platforms [3]. Ethereum, which emerged in 2013, provides a Turing-complete smart contract programming language that supports complex business logic, and smart contracts are gradually being applied in many fields [4]. The enterprise-level consortium chain projects Hyperledger Fabric and FISCO BCOS are underlying blockchain technology platforms, the latter was jointly developed by several Chinese enterprises, both have good support for Turing's complete smart contract language [5, 6]. With the development of smart contracts, some researchers have begun to think that a smart contract is a piece of tamper-proof program code running on the blockchain system. Owing to the short development time of smart contract, it still faces several security problems. In 2016, the Ethereum-based crowd funding project "DAO" was attacked, causing a loss of $60 million [7]. In 2018, the US chain caused a loss of 6 billion RMB owing to the data overflow loophole of the smart contract BatchOverFlow [8]. Mense and Flatscher studied and analyzed 19,366 smart contracts in Ethereum and found that 8,833 of them had security problems [9]. A large number of information security emergencies show that it is urgent and necessary for current smart contract technology to improve on-chain scalability.

The smart contract has the characteristic that it cannot be modified after deployment of the chain, while this ensures the accuracy of program operation, but it increases the difficulty of contract maintenance. Off-chain testing of contracts can only prevent known vulnerabilities, but cannot detect unknown vulnerabilities in the chain. There is no efficient on-chain upgrade for smart contracts up to now. The traditional method is to redeploy the complete contract, which has problems such as destroying the original life cycle of the contract, causing incompatibility between old and new

contract data and affecting normal use of users. The existing three-tier smart contract model includes interface layer, business layer, and data layer. The problem is that the data layer in three-tier contract cannot be upgrade and it also ignores the security of the upgrade on the contract.

This paper studies the upgradeability of smart contracts, optimizes the existing three-tier contract model, redefines the storage layer, abstracts the underlying library table contract in a library table-oriented development method, and implements the unified data storage. This paper adds a verification layer for the contract model, and a transaction rollback mechanism is introduced to verify the correctness and security of the contract while the test data not upload on the chain. According to the model proposed in this study, the smart contract on-chain upgrade algorithm and on-chain verification algorithm are proposed, and the model and algorithm are verified based on the FISCO BCOS platform. The experiment analyzed the model's performance from the aspects of smart contract complexity, deployment cost, on-chain upgrade cost and data migration cost. Contract complexity is a measure of the complexity of the contract's business logic. The contract with lower complexity will has higher execution efficiency. The contract consumption cost is measured by gas; the gas means the consumption of on-chain storage and computing resources by on-chain transactions. The lower the gas consumption, the fewer the on-chain resources are consumed by the contract transactions. The experimental results show that, compared with the traditional contract model, the layered structure model increases the cost of contract deployment and reduces the upgrade cost of the contract chain. Compared with the three-tier contract model, the contract deployment cost of the four-tier smart contract model increases by an average of 19.1%, the complexity of the storage layer contract is reduced by 18.8% on average, and the data migration cost of the storage layer contract is reduced to 0. Therefore, our four-tier smart contract model reduces contract complexity and data migration costs at the cost of some overall deployment.

2. Related Research

Smart contracts not only serve as the carrier of business logic on the chain but also serve as a bridge for users to interact with the blockchain. They are the key technologies for the application of blockchain technology in all walks of life. However, they also present many security risks.

In the field of security detection of smart contracts, in 2018, Sidney et al. proposed a formal verification method using the Isabelle/HOL framework to build smart contract models to verify the security of smart contracts at the bytecode level [10]. Joon-Wie et al. aimed to have safer smart contracts against emerging threats, proposed an approach of sequential learning of smart contract weaknesses using machine learning-long-short term memory (LSTM) to detect new attack trends [11]. In 2019, Zhang et al. proposed a symbolic execution-based method and tool DEFECTCHECKER to detect contract defects that may lead to the bad behavior of smart contracts on the Ethereum blockchain platform [12]. In 2020, Wang et al. proposed ContractWard, an automatic vulnerability detection model for Ethereum smart contracts. This model uses machine learning technology to detect vulnerabilities in smart contracts, extracts binary features from the simplified operation codes of smart contracts, uses five machine learning algorithms and two sampling algorithms to construct the model [13]. Chen et al. proposed an improved smart contract verification tool, Artemis, in 2021, and evaluated the effectiveness and efficiency of the tool's vulnerability detection on 12899 smart contracts [14]. In the area of on-chain governance of block chains, in 2019, Kondapally et al. proposed the idea of using the chameleon hash function to sign blocks, when the current block is signed using the chameleon hash function, once the main verifier agrees to the modification proposal of the block using a digital signature, the current block will be updated [15]. In 2020, Baudlet et al. proposed a new blockchain consensus and governance model that aims to pass the model protection zone, which is not affected by higher-level governance issues and is validated with the existing PoS and PoW [16]. In the same year, Taner et al. designed a decentralized decision-making voting scheme based on identification so that users can implement the on-chain governance of the blockchain by interacting with smart contracts [17]. In 2021, Reijers et al. approach the questions of "on-chain" and "off-chain" governance by reflecting on a long-running debate in legal philosophy regarding the construction of a positivist legal order [18]. In the field of smart contract upgrade methods, in 2020, Angelo et al. proposed a method of using differentiated codes to support smart contract off-chain upgrades by classifying a large number of smart contracts [19]. In the same year, Shao et al. developed a smart contract vulnerability monitoring framework that introduces a log anomaly analysis method into the blockchain log system, transmits the detection results to related contracts, and finally implements prompts for off-chain upgrades of smart contracts [20]. In 2021, Rodler et al. proposed the EVMPatch framework, which can automatically patch defective smart contracts immediately, provide a bytecode rewriting engine for Ethereum, and automatically rewrite common off-the-shelf contracts into upgradeable contracts [21]. In 2022, Zhu and Huang split smart contracts regarding the proxy mode, carried out the implementation and research of the contract upgrade capability, and proposed a smart contract upgrade method under the proxy mode [22].

Security detection, on-chain governance and contract upgrade methods for smart contracts have gradually become a focus of industry research. Breakthroughs in these fields will help solve the security problems of smart contracts. The research on contract upgrade methods mostly focuses on off-chain contract upgrades; the released contracts cannot be quickly upgraded on the chain, and when there is a loophole in the contract, the loss cannot be stopped in time. Therefore, this study uses the upgrade method of the contract chain as the starting point to study the upgrade model of the smart contract chain.

3. Analysis of the Upgrade Problem on the Smart Contract Chain

3.1. Traditional Smart Contract. Due to the tamper-proof feature of blockchain, smart contracts cannot be changed after they are deployed on the chain, so contracts need to be tested extensively to ensure the correctness and security before they are deployed on the chain. The contract testing conducted off-chain can only prevent known vulnerabilities and cannot detect unknown vulnerabilities, such as it cannot detect on-chain vulnerabilities. For smart contracts, there is still no efficient solution to the vulnerabilities that appear after deployment on the chain.

When there are loopholes in smart contracts or changes in demand, the traditional solution is to redeploy a complete contract. This upgrade method has the following problems:

(1) The original life cycle of the smart contract is destroyed, and a new life cycle is restarted after the new contract is deployed

(2) The data of the old and new contracts are incompatible, and the data of the old contract should be migrated

(3) The address of the contract will be updated when redeploying a new contract, affecting the normal use of the user

3.2. Existing Smart Contract Models. In the smart contract chain under the full life-cycle model of smart contracts, there are two ways to achieve partial upgrades:

(1) We use the delegate invocation opcode to forward the function invocation to the target contract. Because the delegate invocation retains the context of the contract and changes the state of the contract, the value of the variables in the contract can be changed by remotely invocating the target contract.

(2) The contract is divided into three layers: proxy, business, and storage. The proxy layer is used as the entry point of the contract invocation, the business layer is responsible for processing the business logic of the contract, and the storage layer is responsible for defining the data structure and data storage.

The three-tier smart contract model structure is loosely coupled. It divides traditional smart contracts into three subsets: entry set, business set, and contract data set. When the contract business needs to be expanded or there are loopholes in the contract, based on the smart contract designed by the three-tier contract model, a low-cost upgrade of the contract business set of the chain can be implemented. The three-tier contract model better solves the problem of local upgrades in the contract chain, but it also has the following two shortcomings:

(1) Only low-cost on-chain upgrades of business logic can be performed, regardless of the upgrade of data-storage-related contracts, that is, low cost on-chain upgrades cannot be performed when storage contracts was changed.

(2) The vulnerabilities of smart contracts often appear in an on-chain production environment after offline testing. The three-tier contract model does not consider the on-chain testing of contracts, that is, the security of on-chain upgrade of smart contracts cannot be guaranteed.

4. Four-Tier Contract Model

4.1. Formal Definition. Based on the three-tier contract model, a four-tier model for smart contracts (hereafter referred to as the four-tier contract model) is proposed. Based on the principle of separation of concerns, this study splits the traditional smart contract into proxy layer, business layer, storage layer, and verification layer, optimizing the storage layer and adding a verification layer compared to the three-tier model, while the remaining two layers basically same as the proxy layer and business layer of the three-tier model. The four-tier contract model is represented by a six-tuple: $\{P, T, L, D, R, G\}$.

The meaning of each element is as follows:

(1) P represents the proxy contract set for the model

(2) T represents the verification contract set of the model

(3) L represents the business contract set in the model

(4) D represents the storage contract set in the model

(5) R represents the correspondence between contract subsets in the model

(6) G represents the upgrade of the contract subset in the model, $G = \{P_g, T_g, L_g, D_g\}$

Let $Z = \{SC, R\}$ and $SC = \{P, T, L, D\}$ denote a complete smart contract. $R = \{(i, j);\ i, j \in \{1, n\}\}$ represents the correspondence between the contract subsets, and it is defined that there are four correspondences including one-to-one, one-to-many, many-to-one, and many-to-many between contract subsets.

The proxy and business contracts both have a one-to-many relationship ($i = 1$, $j = n$). A proxy contract can link multiple business contracts to contract invocations. Only the proxy contract is visible to the user; the proxy contract and the verification contract are a pair of multirelationships ($i = 1$, $j = n$), which are divided into multiple verification contracts according to the principle of single responsibility. The proxy contract can choose partial verification and can also perform process verification on all contracts; the verification contract and the business contract are the same. One-to-one ($i = 1$, $j = 1$) or one-to-many ($i = 1$, $j = n$) relationships can be verified as one-to-one or one-to-many; business contracts and storage contracts are many-to-many ($i = n$, $j = n$) relationships, where each storage contract consists of two parts: the general library table operation and the definition of the data structure.

Given a sequence of smart contract models {Z0, Z1, . . ., Zn−1, Zn}, we define the contract subset upgrade case $\mathrm{Gm} = \{P_g, T_g, L_g, D_g\}$, $g = \{0, 1\}$, where 0 identifies the

contract subset which continues to be in use, 1 identifies the contract subset that needs to be upgraded. The upgrade of any contract subset will not affect the use of other contract subsets. The upgrade of the contract only needs to deploy a new version of the proxy contract and configure the contract addresses of other contract subsets, but it will cause changes to the user's invocation entry, while other contract subsets only need to update the contract link address after redeploying the new version of the contract, and will not affect the user's invocation.

There are four situations for upgrading smart contracts, as shown in Table 1. Case 1 (P_1, T_0, L_0, D_0) will affect user's invocation, which is not in line with the design concept of this model. Therefore, this study only considers the other three cases, that is, the change in the proxy contract is not considered, and the upgrade of the contract will not affect the user, it does not cause any changes.

4.2. Overall Design of the Model. The four-tier contract model was applied to the contract layer of the blockchain architecture. The smart contract, designed based on the four-tier contract model, includes four subsets: proxy, verification, business, and storage contract sets. The application layer interacts with the data layer through contract invocations. The application layer is a decentralized application (DAPP), and the data layer uses blocks and state databases for storage.

The contract subsets of the four-tier contract model are independent of each other, implementing componentization. One layer of the contract is redeployed, and the other layers are not affected. The overall design of the model is shown in Figure 1.

As shown in Figure 1, a user request is forwarded to the contract layer after being classified and processed by the DAPP in the application layer. The contract layer first receives the request from the proxy contract, then the proxy contract invocations the business contract or verification contract, and finally, the storage contract sends the request to the data layer. Blockchain initiates transactions and accesses the blockchain's state database. The contract descriptions of each layer of the four-tier contract model are as follows:

(1) Proxy contract: three main functions are implemented: first, for ordinary requests from users, adapt to the corresponding business contract; second, for upgrade requests from users, update the contract address and link to the new version of the contract; third, to achieve the polymorphism of the proxy contract, call the contract update function that can be linked to a different implementation of the contract.

(2) Business contract: this mainly includes functions such as on-chain business logic processing, transaction definition, and invocation of storage contracts. It implements the transactions of each participant through business functions, tracks the execution status of the transaction, and returns the results to the proxy contract.

(3) Storage contract: it mainly stores data permanently, abstracts this layer in the form of a library table, and invocations it in an SQL-like manner to achieve compatibility between old and new contracts.

(4) Verification contract: this is mainly used to verify upgrade-related information, contract integrity, and detect contact points before the contract upgrade. To ensure the purity of the data on the chain, the test data on the chain is not dropped using assertion rollback and last row rollback.

The two optimizations of the four-tier contract model compared with the three-tier contract model are as follows:

(1) Storage layer optimization: the storage contract was designed as a library table-oriented development method. Based on the precompiled contract of FISCO BCOS, the storage contract and business contract are further decoupled, the visibility of the storage contract is weakened, the chain of the storage contract is implemented, low-cost upgrades and compatibility between old and new contracts achieved.

(2) Add a verification layer to improve the security of on-chain contract upgrades and implement multi-angle on-chain verification of contracts.

The basic process of deploying and upgrading smart contracts based on the four-tier contract model is shown in Figure 2 and described as follows:

(1) After completing the formulation of the smart contract terms, carry out the contract splitting operation, and split the complete contract terms into four contract subsets: proxy, verification, business, and storage contracts.

(2) Test the four contract subsets under the chain. If there is a loophole in the contract, repair the corresponding contract and deploy the smart contract that passes the test on the chain after completion.

(3) When the user needs to upgrade the contract on the chain, invocate the verification contract to verify the new version of the contract on the chain and update the contract address after the verification is passed. Users can upgrade only one layer of contracts without changing other contracts. As long as the proxy contract does not change, the user's invocation is not affected.

(4) Upgraded smart contracts require version management. The business contract set includes a version management contract to save all version information of the contract to trace and manage different versions.

5. On-Chain Upgrade and Verification Algorithm

This section proposes an on-chain upgrade algorithm and a verification algorithm, which are designed according to the proposed four-tier model to implement its main functions.

Table 1: Subset upgrades of contracts.

Case	Proxy contract	Verification contract	Business contract	Storage contract
1: (P_1, T_0, L_0, D_0)	Upgrade	Continue to use	Continue to use	Continue to use
2: (P_0, T_1, L_0, D_0)	Continue to use	Upgrade	Continue to use	Continue to use
3: (P_0, T_0, L_1, D_0)	Continue to use	Continue to use	Upgrade	Continue to use
4: (P_0, T_0, L_0, D_1)	Continue to use	Continue to use	Continue to use	Upgrade

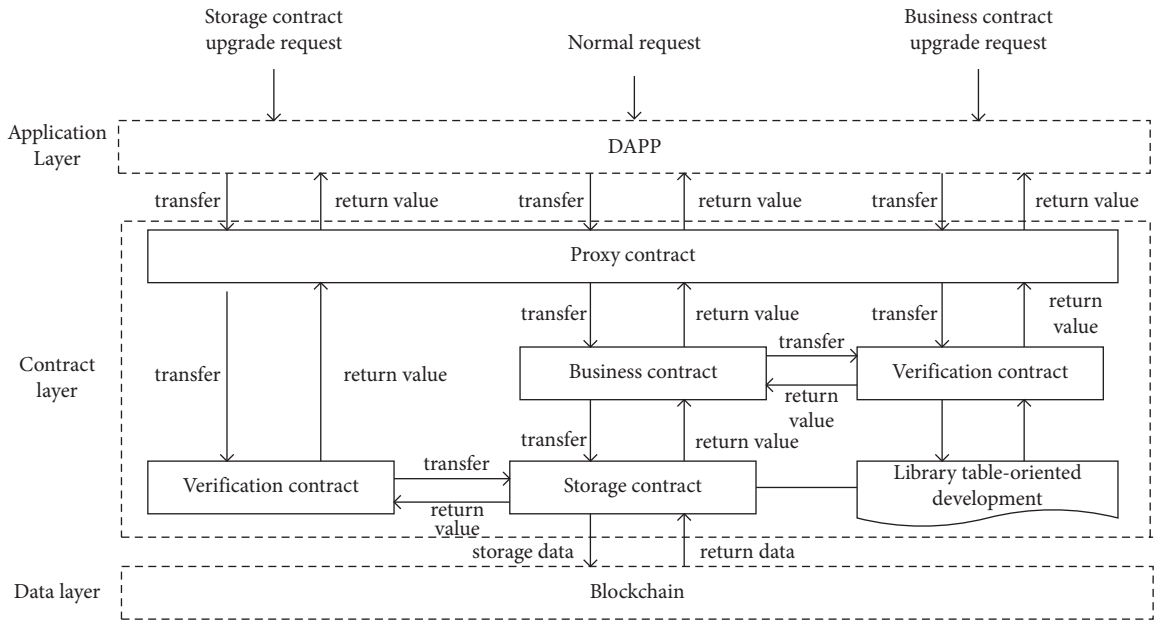

Figure 1: Design diagram of the four-tier model of smart contract.

5.1. On-Chain Upgrade of the Contract. The on-chain upgrade of the contract consists of three steps:

Step 1: User identity verification. Only users with contract upgrade authority can perform contract upgrade-related operations.

Step 2: Verification of the contract chain. The contract can only be updated to the contract subset of the platform after verification.

Step 3: Update the contract's invocation address and destroy the old version of the contract.

The parameters that need to be provided for the on-chain upgrade of the contract are the new version of the contract address, user address, user id, and corresponding operations. Algorithm 1 is the upgrade algorithm of the contract chain, as shown in Algorithm 1.

5.2. On-Chain Verification of Contract. On-chain verification of contracts includes user information verification, contract integrity verification, and contract point detection. User information verification is mainly used to check whether the user has the authority to verify on-chain verification of contracts, which is mainly used to detect necessary methods, such as necessary business function, fallback function, and contract destruction functions. Contract point detection is mainly used for boundary value detection, expected value detection, and abnormal process detection. The parameters that need to be provided for the verification of the contract chain are the contract address, user address, user id, and corresponding operations to be verified. Algorithm 2 is the on-chain verification algorithm for the contract shown in Algorithm 2. The verification of the on-chain contract uses the assertion rollback and last line rollback methods. After the verification succeeds or fails, the transaction will be rolled back to achieve the purpose that the data on the chain is not affected.

6. Reference Implementation and Testing

The model and algorithm proposed in this paper are applicable to current mainstream blockchain platforms, such as FISCO BCOS, Hyperledger Fabric, and Ethereum. This section implements a message interaction system based on the FISCO BCOS platform, implements smart contracts using solidity, and tests and analyzes the four-layer contract model.

6.1. System Deployment and Contract Implementation. In this section, we build a consortium blockchain system consisting of five organizations, five nodes, and three groups, a multigroup parallel method for networking. Where each node of the consortium blockchain in the multigroup parallel network belongs to multiple chains, which can be

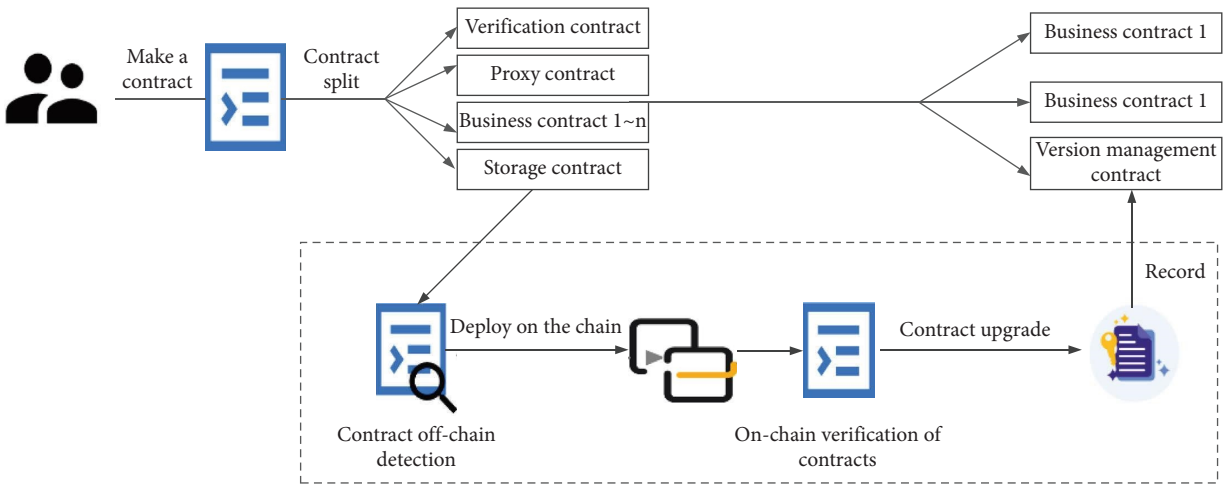

Figure 2: Smart contract deploy and upgrade process.

used for the horizontal expansion of different businesses of multiple parties, or the vertical expansion of the same business. The system contains three chains: the identity chain, information chain 1, and information chain 2. The system network topology is shown in Figure 3.

A smart contract is implemented based on the proposed four-tier model. The business types include information interaction, identity management, and contract version management, and the overall contract is split into proxy contracts, verification contracts, business contracts, and storage contracts. The proxy contract is used as the entry point of the contract invocation for client programs and users to interact with smart contracts. The deployment results are shown in Figure 4.

6.2. Contract Testing and Analysis

6.2.1. Contract Complexity. This section uses the smart contract static analysis tool solidity metric to test contract complexity, and compares and analyzes the complexity of unlayered, three-tier, and four-tier contracts. If the complexity of a contract is lower, the contract's logic will be simpler, and its execution efficiency will be higher. The contract complexity of the business part was tested for the three types of contracts in the system, and the test results are shown in Figure 5.

In Figure 5, the complexities of the unlayered contracts are 167, 124, and 92; the three-tier contracts are 108, 65, and 63; and the four-tier contracts are 108, 85, and 63. It can be seen that the unlayered contract has the highest complexity, and the three- and four-tier structure contracts have the same complexity, indicated that contract hierarchy can reduce the contract's complexity.

In Figure 6, the complexities of the unlayered contracts are 187, 164, and 122, respectively, the complexities of the three-tier contracts are 85, 85, and 85, and the complexities of the four-tier contracts are 69, 69, and 69. It can be seen that the unlayered contract has the highest complexity, followed by the three-tier structure contract, and the four-tier structure contract has the lowest complexity. This shows that the four-tier structure contract reduces the complexity of the storage layer contract.

6.2.2. Contract Deployment Cost. This section tests the overall deployment cost of unlayered, three-tier, and four-tier structured contracts, using gas consumption as a metric. The test results are shown in Figure 7. It can be seen that the more layers a contract has, the higher its deployment cost is. The increase in deployment cost of layered structure contracts is due to the increase in code volume caused by layering and the need for multiple deployments in layers during deployment, which is in line with expectations. The improvement of this model is a way to exchange the deployment cost of the contract for the low-cost upgrade on the contract chain. We have achieved a balance between functionality and performance to some extent, and the increased deployment cost is acceptable compared to the improved upgrade capability on the contract chain.

6.2.3. The On-Chain Upgrade Cost of the Contract. In this section, we test the on-chain upgrade cost of the business part of the unlayered, three-tier structure, and four-tier structure contracts, and the cost is measured by the gas consumption. The test results are shown in Figure 8. It can be seen that the on-chain upgrade cost of the unlayered contract is the highest and increases with the increase of data volume, while the on-chain upgrade cost of the three-tier and four-tier contracts is the same and does not change with the increase of data volume. Therefore, it can be concluded that the layered structure of contracts can effectively reduce the on-chain upgrade cost of the business part of contracts.

6.2.4. Contract Data Migration Cost. In this section, we test the data migration cost due to the upgrade of the storage part of the unlayered contract, the three-tier structure contract and the four-tier structure contract, and the cost is measured in terms of gas consumption. The test results are shown in Figure 9, which shows that the data migration cost increases

```
    Input:
      contractAddr
      userAddr
      userId
      operation
    Output:
      bool
(1)   Begin
(2)     Role ← invokeUserContract (userAddr, userId)
(3)     If (!verifyAuthority (role, operation))
(4)       return false
(5)     ValidationContractInstance ← loadValidationContract ()
(6)     If (!verifyContract (validationContractInstance, contractAddr))
(7)       return false
(8)     ProxyContractInstance ← loadProxyContract()
(9)     UpdateInfo ← updateContractAddr (proxyContractInstance, contractAddr, operation)
(10)    DestroyInfo ← destroyOldContract()
(11)    If (!save (updateInfo, destroyInfo))
(12)      return false
(13)    return true
(14)  End
```

Algorithm 1: Contract on-chain upgrade algorithm.

```
    Input:
      contractAddr
      userAddr
      userId
      operation
    Output:
      bool
(1)   Begin
(2)     Role ← invokeUserContract (userAddr, userId)
(3)     If (!verifyAuthority (role, operation))
(4)       return false
(5)     ContractInstance ← loadContract (contractAddr)
(6)     For (function: fuctionList)
(7)       If (!boundaryValueTest (contractInstance, function))
(8)         revert()//rollback operation
(9)         return false
(10)      If (!expectedTest (contractInstance, function))
(11)        revert()
(12)        return false
(13)      If (!exceptionTest (contractInstance, function))
(14)        revert()
(15)        return false
(16)    revert()
(17)      return true
(18)  End
```

Algorithm 2: Contract verification algorithm.

with the increase of data volume for the unlayered contract and the three-tier structure contract, and the data migration cost for the four-tier structure contract is 0, which is due to the fact that the storage layer of the four-tier structure contract uses the library table-oriented development method to store the data uniformly in the system contract, so the upgrade of the storage layer of the four-tier structure contract does not require data migration.

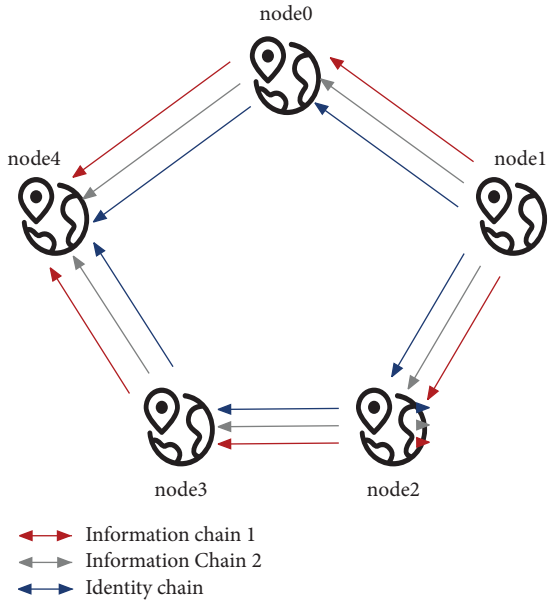

Figure 3: Network topology of the block chain system.

Figure 4: Proxy contract deployment results.

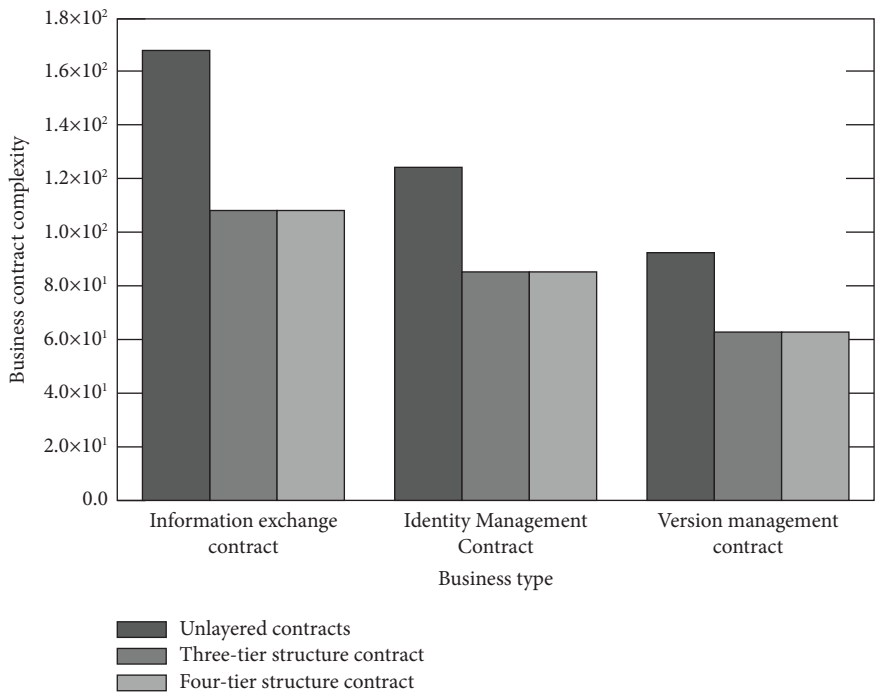

Figure 5: Test result of business contract complexity.

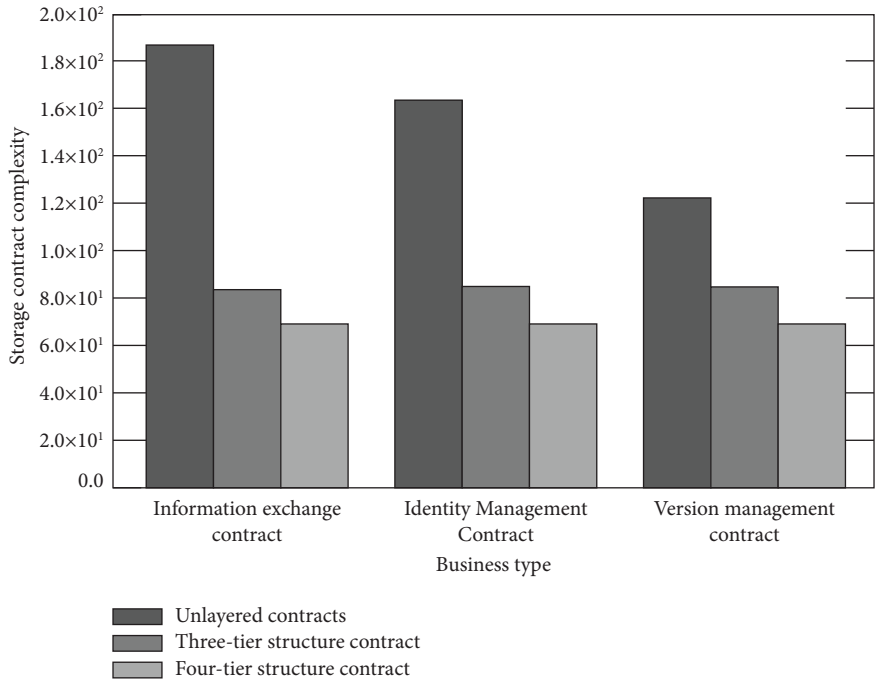

Figure 6: Test result of storage contract complexity.

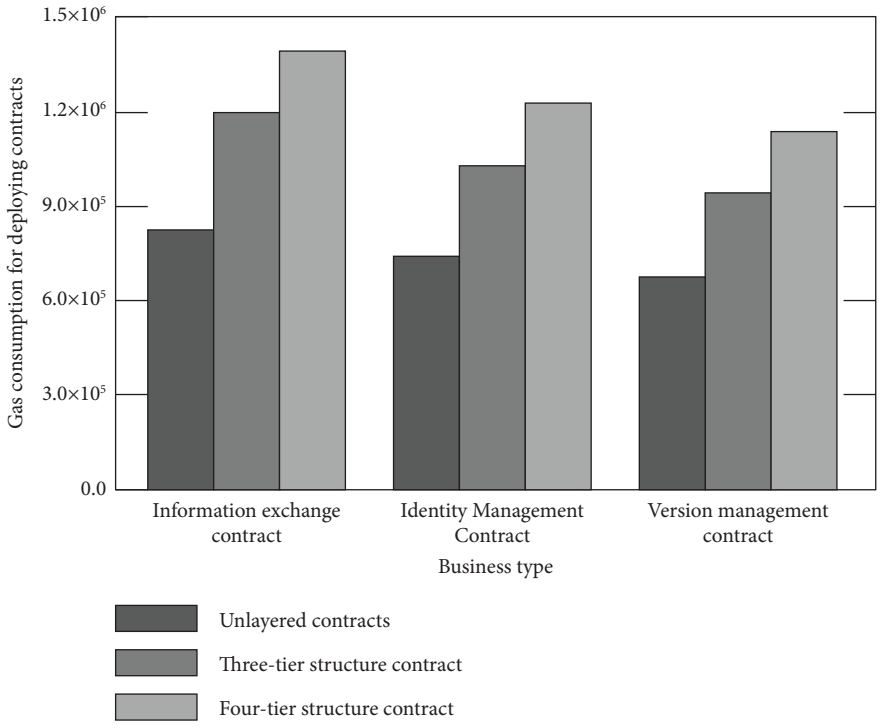

Figure 7: Test result of overall contract deployment cost.

6.3. Brief Summary. In the experiment in this study, compared with the three-tier structure contract model, the deployment cost of smart contracts designed based on the four-tier smart contract model increases by an average of 19.1%, the complexity of the storage layer contract is reduced by an average of 18.8%, and the data migration cost of the storage layer contract is reduced to 0. Therefore, the advantage of the four-tier smart contract model is

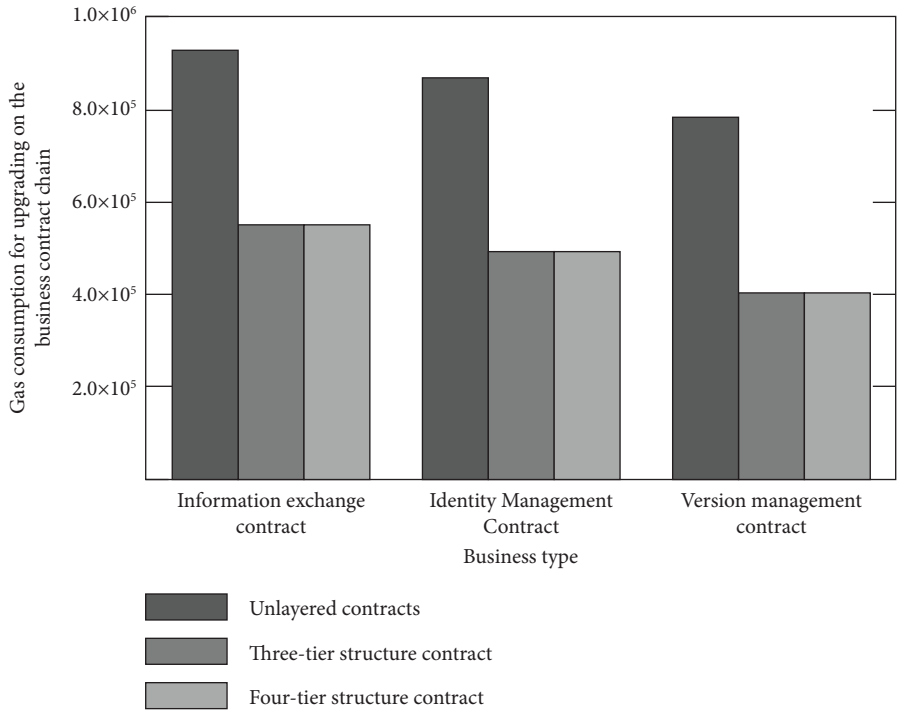

Figure 8: Test result of on-chain upgrade cost of business contracts.

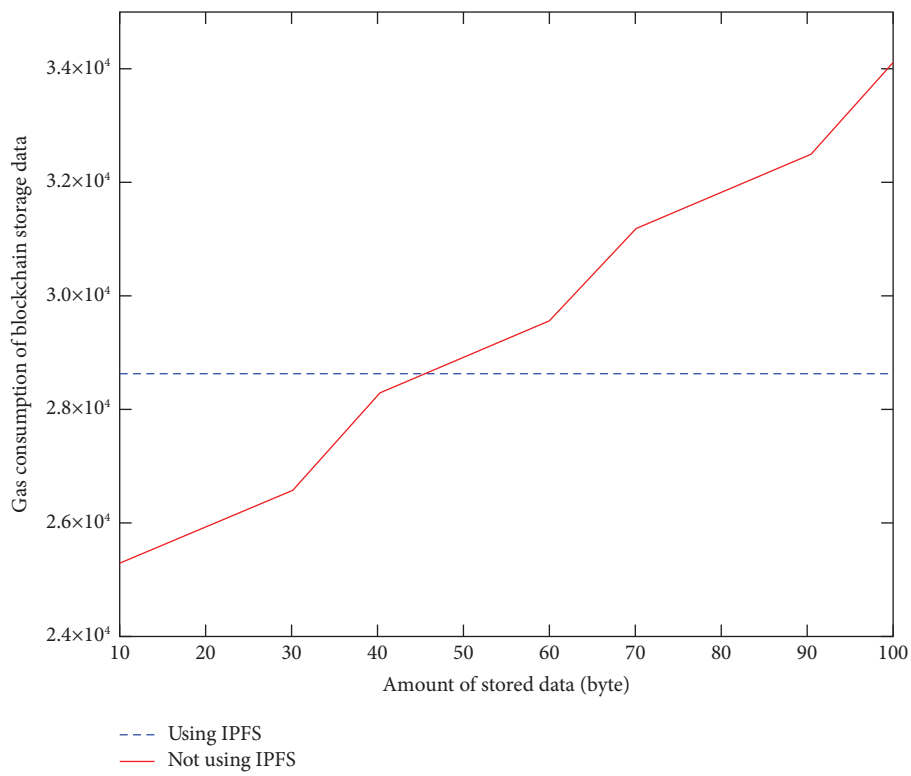

Figure 9: Test result of storage contract data migration cost.

that the complexity of the contract and cost of data migration are reduced. However, the disadvantage is that, with the overall deployment, the cost of the contract increases.

7. Summary and Outlook

In this paper, we propose a four-layer contract model to address the problem of on-chain upgrade of smart contracts. Compared with the three-tier model, the four-tier model adds a verification layer to detect the integrity, boundary value, and abnormal process of the contract and uses the contract rollback mechanism to achieve the purpose of not uploading the test data to the chain; redefine the storage layer to store data uniformly; and achieve the purpose of compatibility of different contract data with the development method oriented to the library table and the proposed upgrade algorithm and verification algorithm on the contract chain, and the on-chain upgrade; Last, the on-chain upgrade, on-chain verification, and version compatibility of the contract are implemented. The four-tier smart contract model reduces the contract's complexity in the storage layer, eliminates the cost of data migration, and increases the contract's overall deployment cost. The cost of increasing the early deployment cost is exchanged for low-cost and flexible upgrades to the later contract. We are going to improve the execution efficiency of the proposed four-layer model, and design and propose an evaluation system for the execution efficiency metrics. In addition, we are currently working on the application and improvement of the four-layer model of smart contracts based on Ethereum and Hyperledger Fabric, etc. We will disclose the relevant experimental data and results in a new research document.

Data Availability

The implementation and testing data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research work was partially supported by the Shaanxi Natural Science Basic Research Project (Grant no. 2020JM-565) and the International Science and Technology Cooperation Program of Shanxi Province (Grant no. 2021KW-07).

References

[1] H. Haiwu, Y. An, and C. Zehua, "Survey of smart contract technology and application based on blockchain [J]," *Journal of Computer Research and Development*, vol. 55, no. 11, p. 2452, 2018.

[2] S. Wang, Y. Yuan, X. Wang, L. Juanjuan, and Q. Rui, "An overview of smart contract: architecture, applications, and future trends [C]," in *Proceedings of the 2018 IEEE Intelligent Vehicles Symposium (IV) IEEE*, pp. 108–113, Changshu, China, June 2018.

[3] P. Dixit, A. Bansal, P. S. Rathore, and M. Payal, "Blockchain technology and the internet of things," *An Overview of Blockchain Technology: Architecture, Consensus Algorithm, and its Challenges*, pp. 21–46, Apple Academic Press, Marie, Canada, 2020.

[4] A. Mavridou and A. Laszka, "Designing secure ethereum smart contracts: a finite state machine based approach [C]," in *Proceedings of the International Conference on Financial Cryptography and Data Security*, pp. 523–540, Springer, Berlin, Germany, February 2018.

[5] K. Yamashita, Y. Nomura, E. Zhou, B. Pi, and S. Jun, "Potential risks of hyperledger fabric smart contracts [C]," in *Proceedings of the 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE).IEEE*, pp. 1–10, Hangzhou, China, March 2019.

[6] Z. Hui, X. Chen, and X. Hao, "An overview on practice of FISCO BCOS technology and application [J]," *Information and Communications Techno-logy and Policy*, vol. 46, no. 1, p. 52, 2020.

[7] M. I. Mehar, C. L. Shier, A. Giambattista et al., "Understanding a revolutionary and flawed grand experiment in blockchain: the DAO attack [J]," *Journal of Cases on Information Technology*, vol. 21, no. 1, pp. 19–32, 2019.

[8] Y. Feng, E. Torlak, and R. Bodik, "Precise attack synthesis for smart contracts," 2019, https://arxiv.org/abs/1902.06067.

[9] A. Mense and M. Flatscher, "Security vulnerabilities in ethereum smart contracts [C]," in *Proceedings of the 20th International Conference on Information Integration-N and Web-Based Applications & Services*, pp. 375–380, Halifax, NS, USA, August 2018.

[10] S. Amani, M. Begel, M. Bortin, and M. Staples, "Towards verifying ethereum smart contract byte code in isabelle/HOL [C]," in *Proceedings of the 7th ACM SIGPLAN Inter-national Conference on Certified Programs and Proofs*, pp. 66–77, Los Angeles, CA, USA, January 2018.

[11] W. Joon-Wie Tann, X. Jie Han, S. S. Gupta, and S. Y. Ong, "Towards safer smart contracts: a sequence learning approach to detecting security threats [J]," 2018, https://arxiv.org/abs/1811.06632.

[12] P. Zhang, F. Xiao, and X. Luo, "Solidity check: quickly detecting smart contract problems through regular expressions [J]," 2019, https://arxiv.org/abs/1911.09425.

[13] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: automated vulnerability detection models for ethereum smart contracts," *IEEE Transactions on Network Science and Engineering*, vol. 8, no. 2, pp. 1133–1144, 2021.

[14] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "Defectchecker: automated smart contract defect detection by analyzing EVM bytecode [J]," *IEEE Transactions on Software Engineering*, vol. 48, no. 7, 2021.

[15] K. Ashritha, M. Sindhu, and K. V. Lakshmy, "Redactable blockchain using enhanced chameleon hash function [C]," in *Proceedings of the 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS)*, pp. 323–328, Coimbatore, India, March 2019.

[16] M. Baudlet, F. Doudou, Y. Taenaka, and Y. Kadobayashi, "The best of both worlds: a new composite framework leveraging pos and pow for blockchain security and governance [C]," in *Proceedings of the 2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pp. 17–24, Paris, France, September 2020.

[17] T. Dursun and B. B. Ustundag, "A novel framework for policy based on-chain governance of blockchain networks,"

Information Processing & Management, vol. 58, no. 4, Article ID 102556, 2021.

[18] W. Reijers, I. Wuisman, M. Mannan et al., "Now the code runs itself: on-chain and off-chain governance of blockchain technologies," *Topoi*, vol. 40, no. 4, pp. 821–831, 2021.

[19] M. Angelo and G. Salzer, "Characterizing types of smart contracts in the ethereum landscape [C]," in *International Conference on Financial Cryptography and Data Security*, pp. 389–404, Kota Kinabalu, Malaysia, March 2020.

[20] W. Shao, Z. Wang, X. Wang, K. Qiu, C. Jia, and C. Jiang, "LSC: online auto-update smart contracts for fortifying blockchain based log systems," *Information Sciences*, vol. 512, pp. 506–517, 2020.

[21] M. Rodler, W. Li, G. O. Karame, and L. Davi, "EVMPatch: Timely and Automated Patching of Ethereum Smart Contracts [C]," in *Proceedings of the 30th USENIX Security Symposium (USENIX Security 21)*, pp. 1289–1306, August 2021.

[22] Y. Zhu and J. Huang, "Research on how to realize the upgrade-ability of smart contracts in the proxy mode [J]," *Journal of Harbin Institute of Technology*, vol. 54, no. 2, pp. 267–272, 2022.