IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 19, 2022, accepted May 5, 2022, date of publication May 10, 2022, date of current version May 16, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3174052

A Review on Recent Progress of Smart Contract in Blockchain

CANGHAI WU, JIE XIONG, HUANLIANG XIONG, YINGDING ZHAO, AND WENLONG YI
Software College, Jiangxi Agricultural University, Nanchang 330045, China

Corresponding author: Huanliang Xiong (xionghuanliang@jxau.edu.cn)

This work was supported in part by the National Key Research and Development Projects under Grant 2020YFD1100605, in part by the National Nature Fund Project under Grant 61762048, in part by the Scientific Research Project of Jiangxi Provincial Department of Education under Grant GJJ190180 and Grant GJJ200428, and in part by the Scientific Research Fund Project of Jiangxi Agricultural University under Grant 9232307210.

ABSTRACT A smart contract, in form, is represented as a piece of computer program code involving related commercial transactions and algorithms. Essentially, this is the computerization of the pre-agreed contract between the participants. This special contract agreement is automatically verified and executed once preset conditions are triggered. Smart contracts are not only used in the field of financial transactions, but also include many aspects of social life. Although smart contract technology has unique advantages, it is still in the early stages of development, and many problems remain to be solved. First, this article briefly summarizes the development process of blockchain, and then focuses on the research progress of blockchain 2.0-smart contracts. Second, the related concepts of smart contracts are presented, and the working mechanism of smart contracts and the difficulties faced by smart contracts are elaborated. Finally, in response to these problems and dilemmas, the corresponding solutions and ideas are summarized, and the future challenges and development trends of smart contracts are analyzed and judged.

INDEX TERMS Blockchain, smart contract, transaction automation, non tamperability.

I. INTRODUCTION

In 2009, Bitcoin [1] with the characteristics of decentralization and anonymity was born, which has attracted widespread attention from people in various fields, especially experts and scholars in the computer field, and its key technology, blockchain [2], has also been well known. Blockchain is a decentralized distributed ledger system, which is essentially a distributed database that incorporates technologies such as asymmetric encryption algorithms, point-to-point transmission, consensus mechanisms, and smart contracts, which can ensure the privacy of users during transactions, and solve the problem that the third-party platform cannot be trusted.

The development of blockchain technology can be divided into three stages: 1) The programmable currency stage of blockchain 1.0 [3], during which, a large number of digital currencies represented by Bitcoin appeared. 2) The programmable financial stage of blockchain 2.0, the most representative system is the Ethereum [4], which provides a Turing-complete smart contract system, allowing developers to program any decentralized application. 3) The programmable society stage of blockchain 3.0, which goes beyond the ordinary economic and social fields and provides decentralized solutions [5] for the ''programmable society''. At this stage, the blockchain has passed the 1.0 stage and is in the process of moving towards blockchain 2.0, that is, the smart contract stage.

The key to the success of the Ethereum system is the adoption of smart-contract technology. Smart contracts were first proposed by Szabo [6] in 1995, emphasizing that they facilitate the execution of contracts through the use of protocols and user interfaces. A smart contract is deployed to operate in the blockchain. When predetermined conditions are satisfied, it is triggered to execute and update the blockchain. Once a contract related to the transaction is executed, the transaction result cannot be changed or reversed. Smart contracts work similar to If-Then statements [7] in other computer programs, and in this way, smart contracts interact with real-world assets.

This article consists of six sections: 1) A brief introduction to the blockchain development process and its current stage. 2) An overview of the basic concepts of smart contracts, including the development process of blockchain technology,

The associate editor coordinating the review of this manuscript and approving it for publication was Yinliang Xu.

basic concepts of smart contracts, smart contract working mechanisms, and the dilemma of smart contracts. 3) Detailed analysis of the problems in the progress of smart contracts, including performance, privacy, and security issues of smart contracts. 4) In-depth summary and commentary on solutions to key issues in smart contracts. 5) Analysis and judgment of the challenges faced by smart contracts in the future as well as the development trend of smart contracts. 6) Summary of full text.

II. BASIC CONCEPTS OF SMART CONTRACTS

A. INTRODUCTION TO BLOCKCHAIN

Blockchain is a chain structure formed by the orderly concatenation of data blocks according to the generation time, and is also a distributed database [8] with the characteristics of decentralization, collective maintenance, tamper proof, and distrust, and is especially suitable for building a programmable money system. Blockchain technology solves two important problems for Bitcoin platform: namely the dual payment problem and the Byzantine Generals Problem [9]. The dual payment problem, also known as ''double-spending,'' uses the digital properties of money to complete a payment with the ''same money'' two or more times. Blockchain technology can solve the dual-payment problem of a decentralized system through the verification and consensus mechanism of distributed nodes without a third-party organization, and complete the value transfer in the process of information transmission. The Byzantine Generals Problem is a common problem faced in the interaction process of distributed systems that is, in the absence of a trusted central node, how distributed nodes reach consensus and establish mutual trust [10]. Blockchain uses digital encryption and distributed consensus algorithms to build a decentralized trusted system without the need for trusted individual nodes. In contrast to the credit endorsement mechanism of traditional central institutions (such as central banks), the Bitcoin system forms software-defined credit [11], which marks a fundamental change from centralized national credit to decentralized algorithmic credit.

Since the advent of blockchain, it has presented broad application prospects and attracted considerable attention from academia and industry. Blockchain technology has been widely used in medical care, finance, Internet of Things, energy, and many other fields. Blockchain can generally be divided into public blockchain, consortium blockchain and private blockchain according to the access permission. Public blockchains are open to all users in the world, so any user can read data and broadcast transactions on the chain. The consortium blockchains are jointly managed by several business-related institutions, each of which runs one or more nodes, and the read-write permissions are limited to the nodes in the consortium. The read-write permissions of the private blockchains are controlled by an organization or institution, and the qualifications of participating nodes are strictly limited [12].

Since 2016, smart contract technology represented by Ethereum has become the focus of attention from all walks of life, attracting extensive attention from government departments, financial institutions, and technology companies [13]. In December 2016, the first Smart Contract Symposium was held at Microsoft's New York City headquarters to, analyze and discuss the application scenarios of smart contracts. In February 2017, the European Parliament published the report ''How Blockchain Changes Our Lives'' [14], indicating that smart contract technology is the most promising blockchain application. The Enterprise Ethereum Alliance (EEA) was established in the same month. EEA is committed to developing Ethereum into an enterprise-level blockchain. Its members include large financial institutions such as JPMorgan Chase and ABN Amro, as well as famous technology companies such as Microsoft and Intel.

Blockchain includes technologies such as distributed architectures, consensus algorithms, and smart contracts. Smart contract technology can ensure that users who do not trust each other complete transactions without any third-party trusted intermediaries or authorities. Simultaneously, smart contracts in digital form can be flexibly embedded in various tangible or intangible assets, transactions, and data to realize active or passive assets, information management and control, and gradually build programmable smart assets, systems, and society.

B. OVERVIEW OF SMART CONTRACTS

Definition 1 (Smart Contract): Let the contract drawn up by the transaction parties $u_1, u_2 \ldots, u_k (k \epsilon Z^+)$ in the real world be C, the smart contract is recorded as IC, the trusted third-party institution is G, and under the supervision of the institution G, the parties to the contract, the result of performing the contract C is recorded as R, that is, $R = C(U, G)$, then $R = IC(U)$, $\mathrm{U} = \{u_1, u_2, \ldots, u_\mathrm{k}\}$. Smart contracts automatically complete transaction contracts that require the supervision of a trusted third-party organization in the real world to ensure that the contracts are actually fulfilled.

A smart contract defines a set of commitments in the form of digitals [15]. The smart contract in the Ethereum system is a program control protocol for digital currency assets based on blockchain technology [16]. From the perspective of a computer, a smart contract can be understood as a piece of computer program code that involves related commercial transactions and algorithms. From the public perspective, a smart contract is a related agreement. Once the contract preset conditions are triggered, the smart contract is automatically validated and executed. Smart contracts are not only used in the field of financial transactions, but also in many other aspects of social life, such as agricultural scientific and technological achievements, the Internet of Things and other fields.

Smart contracts have a life cycle, which includes three stages: contract generation, contract release, and contract execution, as shown in Figure 1.

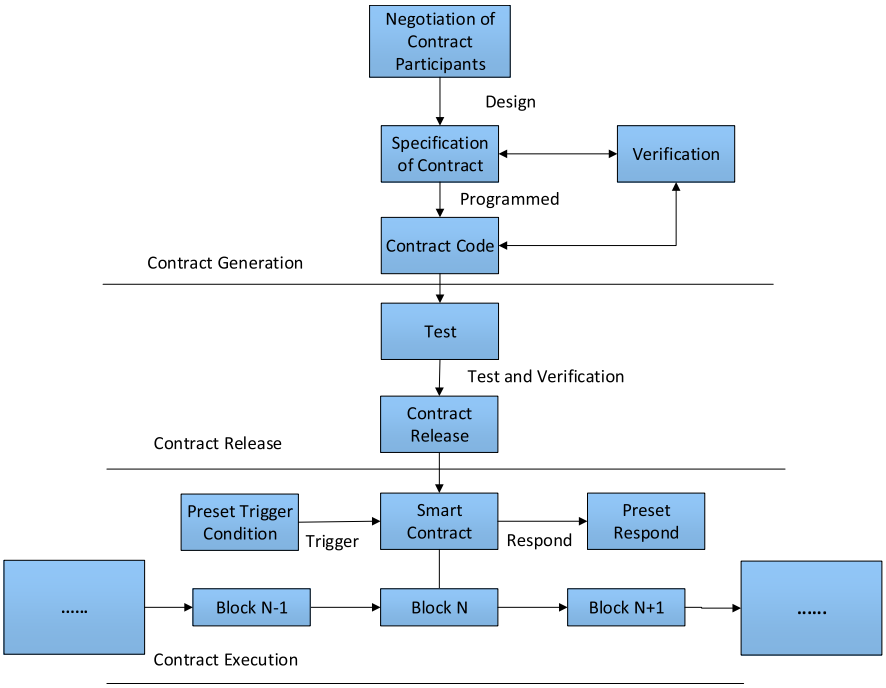

FIGURE 1. Smart contract life cycle.

Contract generation includes several steps: contract multi-party negotiation, contract specification formulations, contract verification, and obtaining contract code. First, the parties involved in the contract negotiate to clarify each other's rights and obligations, determine the standard contract text and program it, and obtain the standard contract code after verification. The contract generation process involves two important links: the contract specification and contract verification. Contract specifications need to be negotiated and formulated by experts with relevant domain knowledge and contract parties. Contract verification is carried out on a virtual machine based on the system abstract model, which is related to the security of the contract execution process, and the consistency of the contract code and the contract text must be guaranteed.

After the contract is generated, the next stage is contract release. The signed contracts are distributed to the nodes in P2P mode, and the node temporarily stores the received contract in memory and waits for a consensus to be reached.

The main steps of the consensus process include:

(1) Package for contract collection. Each node package temporarily stored contracts in the recent period to form a contract set.
(2) Contract blocks are generated and broadcast to the entire network. Calculate the hash value of all contracts in the contract set and then assemble these hash values into a new block and publish it to other nodes in the entire network.
(3) Other nodes validate blocks. After receiving the newly broadcasted block, other nodes compare the hash value in the block with the Hash value of the contract set saved for verification.
(4) Multiple rounds of comparison, consensus reached, and the entire network broadcast. After several rounds of sending and comparison, all nodes eventually reach a consensus on the newly released contract, and the consensus set of contracts is broadcast to all nodes in the entire network in the form of blocks.

Once a contract is released, it cannot be changed. If the preset conditions are satisfied, the contract is automatically executed. Contract execution is based on the "event trigger" mechanism. The smart contract subsystem in the blockchain system has transaction processing and preservation functions, as well as a complete state machine for accepting and processing various smart contracts. The smart contract subsystem periodically traverses the state machine and trigger conditions of each contract, and pushes contracts that meet the trigger conditions to the queue to be verified. The contracts to be verified are broadcast to each node. Similar to ordinary blockchain transactions, the node first performs signature verification to ensure contract validity. The verified contract will be successfully executed after reaching a consensus. The entire contract processing process is automatically completed by the smart contract subsystem built into the bottom layer of the blockchain, which is open and transparent, and cannot be tampered with.

In essence, the realization of a smart contract is to give the object digital characteristics: that is, the object is programmed and deployed on the blockchain to become a resource shared by the whole network, and then trigger the automatic

generation and execution of the contract through external events, so as to change the state and value of digital objects in the blockchain network. Existing smart contract technology platforms, such as Ethereum and Hyperledger, have a Turing-complete script development language, enabling blockchain to support more smart contract applications in the financial and social system fields.

Although smart contracts are not widely used, their significant advantages have been recognized by many industry researchers. Compared with traditional contracts, they have the following significant advantages:

1) Reducing transaction risks. Owing to the immutable nature of blockchain, smart contracts cannot be changed at will once they are released on the chain. Furthermore, all transactions stored and replicated throughout the distributed blockchain system are traceable and auditable. Thus, malicious acts like financial fraud can be greatly mitigated.
2) Reducing administrative and service costs. Blockchain ensures the trustworthiness of the entire system through a distributed consensus mechanism without going through a central broker or intermediary. Once the smart contract stored in any block is triggered, it is broadcast to the entire blockchain network after being verified and executed by the nodes. As a result, administrative and service costs can be significantly reduced by eliminating the need for third-party intervention.
3) Improving the efficiency of business processes. Removing dependency on mediation can significantly improve the efficiency of business processes. For example, once predefined commodity supply chain procedures are met, such as the buyer confirming receipt of the relevant product, financial settlement will be automatically completed in a point-to-point manner, thereby greatly shortening the transaction turnaround time.

C. WORKING MECHANISM OF SMART CONTRACTS

The smart contract contains two attributes: the state variable and state value. In the smart contract program, If-Then and What-If statements are used to set the triggering scenarios and response rules of the terms in the contract. Through multi-party mutual agreement and digital signature, the user submits the transaction initiated. After propagation through the blockchain network and verification by each node, it is stored in blocks of the blockchain. The user obtains the contract address and contract interface, and invokes the contract during trading. Miners accept the incentive mechanism set by the system, contribute their computing power to verify transactions, and generate contracts or execute contract codes in the local sandbox after receiving the contract creation or invocation command. The contract codes automatically determine whether the current scenario meets the contract trigger conditions to strictly implement the response rules and update the world state. After the transaction is verified to be valid, it is packaged into a new data block, which is linked to the main chain of the blockchain after consensus authentication [17].

Owing to the differences in blockchain platforms and their operating mechanisms, and the differencees in smart contract development languages, the operating mechanisms of smart contracts are also different. Therefore, the working mechanism of the smart contract is explained from the three common aspects of the smart contract subject, the data loading method, and the execution environment.

1) CONTRACT SUBJECT

1) A smart contract body is a complex protocol framework for contract applications that can identify the behavior and state of the contract.. The smart contract body includes two main parts: protocol and module parameters. The main structure of smart contract is shown in Figure 2
2) The protocol is a procedural description of the legal text issued by the standard organization. [18]. The protocol includes legal standard text and standard parameters, each of which has an identifier that represents a type. Therefore, a protocol can be considered a fully instantiated template.
3) Module parameters include the main parameters of the smart contract and various accessory modules. The business logic module includes customized legal texts and parameters. This is a programmatic description of the professional knowledge in the field of application. It is negotiated by the contract participants and involves the rights and obligations of multiple parties. All parameters are critical parts of the contract, as they not only directly reflect the business relationship between the parties but also affect the automatic execution of the contract.

2) DATA STORAGE AND LOADING METHODS

The main data of smart contracts include status data, transaction data, contract code, and application data etc. Status and transaction data are generally stored online for easy observation and verification. The contract code and application data are divided into two loading methods: on -chain and off - chain. Most blockchain systems use the on -chain storage mode, that is, code and application data are sent to the blockchain, and then the data and code are loaded from the chain and executed. The disadvantage is that it occupies the storage resources of the node and accumulates data for a long time, resulting in a huge storage burden. The off-chain approach stores the hashes of smart contracts on the chain, and then stores the complete contract code in a storage network or trusted database indexed by hashes, such as the IPFS system and the Tower Crier platform [19]. The hash value is calculated based on the content of the contract code to ensure immutability of the contract. The hash index method adopted by a blockchain system can save considerable storage

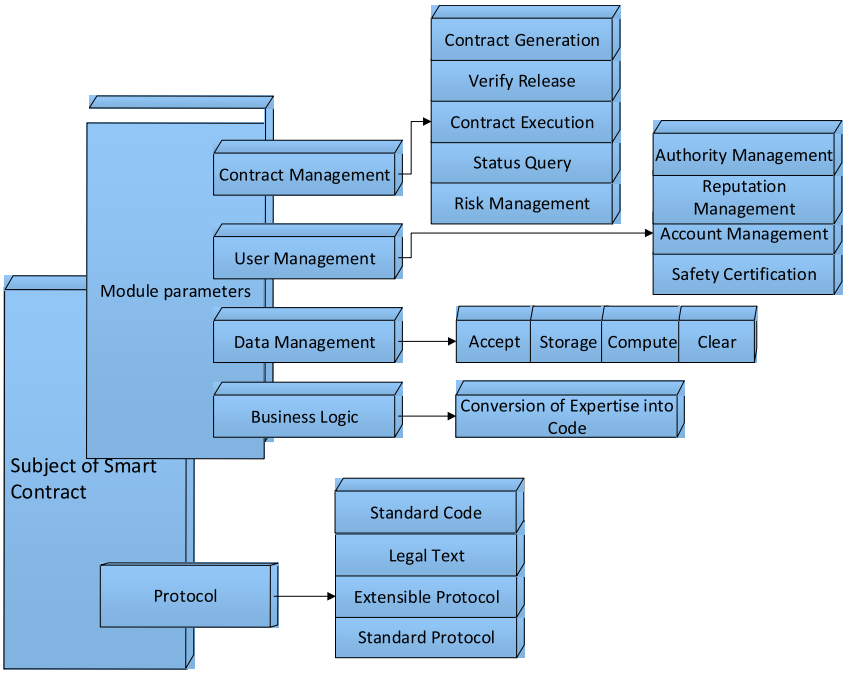

FIGURE 2. The main structure of the smart contract.

space. Simultaneously, the invisible contract code strengthens the protection of contract privacy.

3) THE EXECUTION ENVIRONMENT OF THE CONTRACT

At present, there are two main types of execution environments for smart contracts: virtual machines and containers (dockers). The virtual machine and container are similar to a sandbox that isolates and limits the resources used by the contract while executing the contract code. A virtual machine usually refers to the software implementation of a computer with complete hardware functions that can execute programs like a real machine, and is a computer simulated by software, such as VMware. To reduce resource overhead and improve performance, most blockchains use lightweight virtual machine structures, such as the Ethereum Virtual Machine (EVM). An EVM flowchart is shown in Figure 3.

Containers are kernel virtualization technologies that provide lightweight virtualization to isolate processes and resources. In the Linux operating system, containers are typically created by Docker, which isolates the external environment and provides an independent running environment for smart contracts. Hyperledger Fabric uses lightweight Docker as a smart contract execution environment. Docker uses a sandbox mechanism with no interfaces. The program code in Docker runs directly on the underlying operating system, and its execution efficiency is very high. The running process of the Docker is shown in Figure 4.

4) CONTRACT CREATION AND EXECUTION PROCESS

In ethereum system, contract creation can be regarded as a special transaction process. The contract creation function uses a set of fixed parameters to create a new contract and generate a new set of states. The process is as follows:

$$(\sigma', g', A) \equiv \Lambda(\sigma s, o, g, p, v, i, e)$$

where σ is the system status, $s$is the transaction sender, o is the transaction source account subject, g is the available gas, p is the gas price, v is the account amount, i is the initial EVM code, and e is the depth of creating the contract stack. σ' is the new state of the system, g' is the residual gas, and A is the sub-state. Finally, by performing the initialization EVM code, a new contract account is created, and the account address, storage space and the main code of the account are generated. In this process, excluding the gas consumed by the transaction that occurs, the gas consumption of code creation is proportional to the amount of code in the contract created. However, once the gas surplus is less than the gas required for code creation, a GAS exception (OOG) is generated, and the gas surplus is set to zero and no new contracts are created.

The contract operation model describes how the state of the system transitions after receiving a set of bytecode and environment data tuples. In practice, the model consists of the iterative process of the whole system state and the virtual machine state. The iterator continuously runs the iteration function until the virtual machine is paused by an abnormal state (OOG) or by producing normal result data. The smart contract deployment flow chart is shown in Figure 5.

Usually, intelligent encapsulation predefined number of contract status, transformation rules, the trigger condition, and deal with, such as the operation after signed by the parties

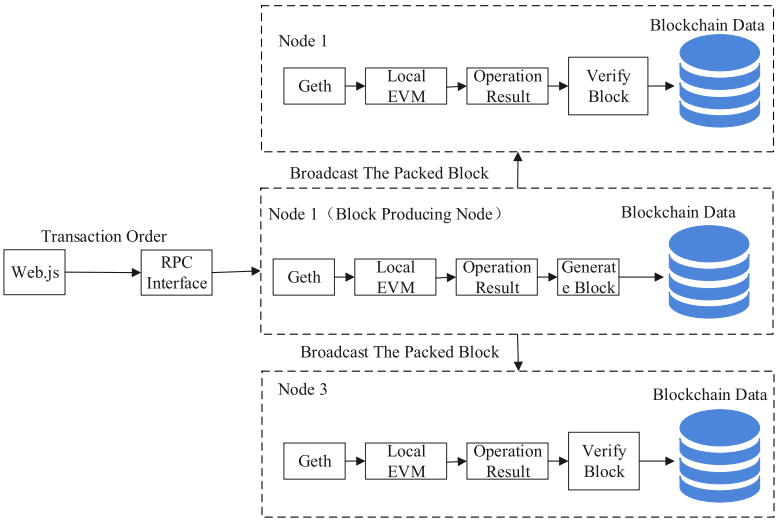

FIGURE 3. EVM operation flow chart.

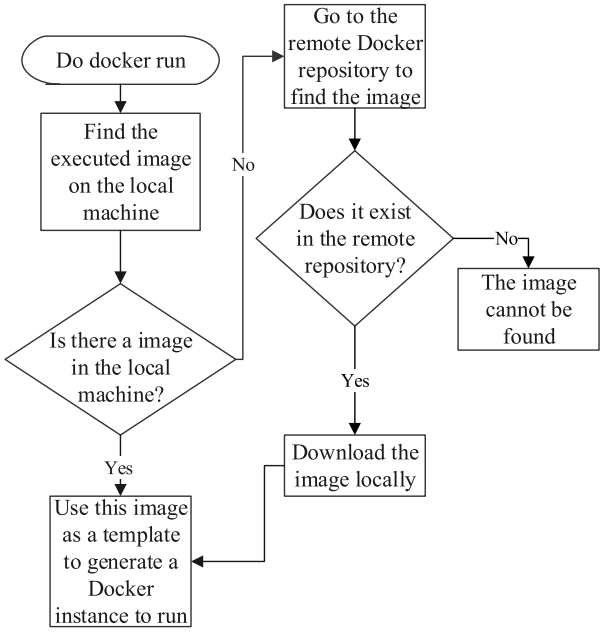

FIGURE 4. Docker running flow chart.

in the form of program code attached on the data block chain, transmission through P2P networks and nodes in each node of the distributed books after verification, block chain can real-time monitoring the whole intelligent contract status, The contract is activated and executed after the external data source is verified to meet specific trigger conditions. The operation flow chart of the smart contract is shown in Figure 6.

D. THE DILEMMA OF SMART CONTRACTS

Although smart contract technology has unique advantages, it is still in the initial stage of development, and many problems remain to be overcome, as listed in Table 1.

(1) In terms of basic performance, smart contracts have problems such as inefficient contract execution and difficulty in expanding contract data storage.
(2) In terms of data privacy, smart contracts are prone to disclosure, mainly including trusted data source privacy and contract data privacy disclosure, which are closely related to the infrastructure and the contract layers.
(3) In terms of security, smart contracts are vulnerable to potential security vulnerabilities, such as the operating environment, compilation process and program characteristics of smart contracts. Smart contracts are decentralized applications running on the blockchain, and their security of smart contracts is largely subject to the security of their operating environment. Smart contracts written in high-level languages are compiled into byte codes and executed by the transaction drivers. These program features and the compilation execution process also pose potential security threats. Therefore, the security threats of smart contracts mainly come from potential vulnerabilities at the three levels of virtual machines, high-level languages, and blockchains.

III. KEY ISSUES OF SMART CONTRACTS

A. INEFFICIENT EXECUTION OF SMART CONTRACTS

The traditional blockchain adopts a single-chain data structure, outside the genesis block, where each block has only one predecessor block, and the blocks are serially connected by hash pointers in the order of block production time to form a single chain [20]. In this blockchain system, smart contracts are executed serially in chronological order, and it takes a long waiting time until the contract is executed, which results in a very limited number of contracts executed per second by the system. Moreover, the distributed single-chain structure of the blockchain system limits the execution

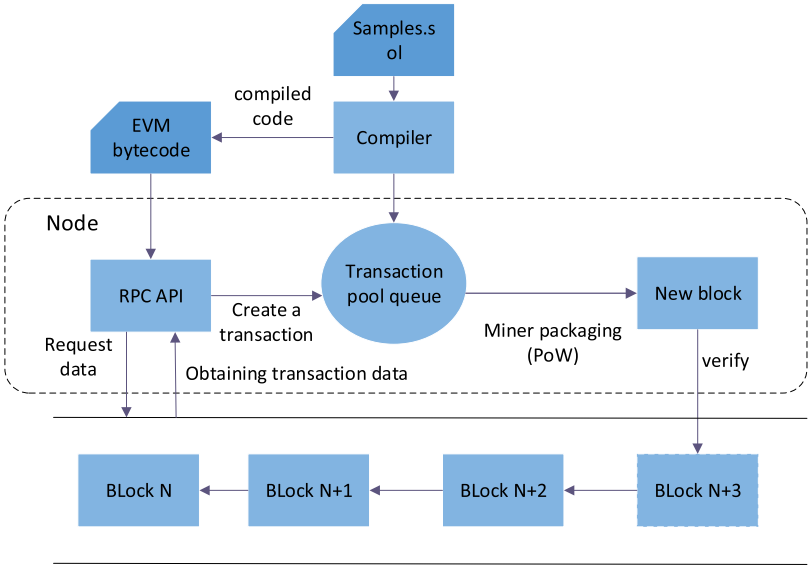

FIGURE 5. The smart contract deployment flow chart.

TABLE 1. Problems with blockchain smart contracts.

Type / Feature	Performance issues	Privacy issues	Security issues
Content	Inefficient contract execution and difficulty in expanding contract data storage	Trusted data source privacy and contract data privacy disclosure	Smart contracts are vulnerable to potential security vulnerabilities, such as the operating environment, compilation process and program of characteristics smart contracts
Threat, attack, challenge	Low throughput, data storage difficulties, and poor scalability	Artificial steal, control network nodes, profit chain code vulnerability to obtain private information	Cause huge economic losses, leak user information, contract loopholes difficult to repair
Solutions	Concurrent execution, off-chain computing, and contract microservices	Channel isolation, power limit system, can be hard Component execution environment	Fuzzing testing, symbolic execution, formal verification and other technologies.

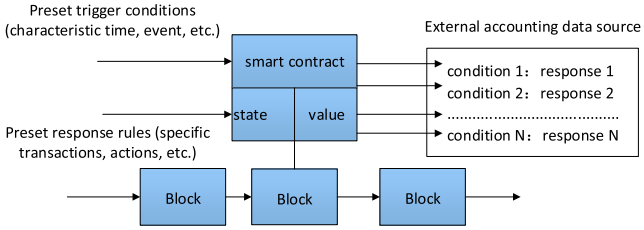

FIGURE 6. The operation flow chart of the smart contract.

efficiency of contracts to a single node, which is incompatible with popular multicore and cluster architectures, resulting in low execution efficiency of blockchain smart contracts and difficulty in meeting the needs of multifield applications [21].

For example, the Ethereum system currently processes an extremely limited number of transactions per second, approximately 10 to 20, however, to complete the confirmation of these transactions, it needs to wait until the next block is generated, and a new block is generated in Ethereum for approximately 12 s, which limits the number of transactions that Ethereum can confirm each time, and a large number of contract transactions cannot be processed and confirmed in time. Therefore, improving the execution efficiency of smart contracts such that the blockchain system can process more transactions per unit time is very important and urgent.

B. CONTRACT DATA STORAGE IS DIFFICULT TO SCALE

The smart contract blockchain records all the state changes of the blockchain network from its birth to the current moment, and requires each node to maintain a complete data backup. In fact, these massive amounts of contract data continue to grow rapidly. For nodes in the chain, it is extremely difficult to store and synchronize these increasingly large amounts of data, and the contract data storage is difficult to expand.

For example, in the Ethereum system, more than 1TB of storage space is required to synchronize all block data fully after the genesis block. For new nodes entering the network, fully synchronizing blockchain data not only takes

up a huge storage space, but also requires more than two weeks of synchronization time. This long synchronization period is almost unacceptable. The data in the Ethereum system includes transaction data and the smart contract code itself, among which the total amount of smart contract code data accounts for a considerable proportion. Even if the smart contract is chained separately, according to the increasingly active trend of Ethereum and the cumulative effect of time, its huge amount of smart contract data is also a problem that needs to be solved urgently, and it is urgent to explore and seek scalable storage solutions for smart contract data.

1) PRIVACY ISSUES OF TRUSTED DATA SOURCES

After applying smart contracts to a blockchain system, the application scenarios of the blockchain become more abundant and the operating environment becomes more complex. External trusted data sources may need to be queried during smart contract execution. These request operations are typically public available. If such public information is illegally obtained, user privacy will undoubtedly be leaked. For example, when a user initiates a transaction, the client calls the corresponding transaction function, and the system creates a smart contract for the transaction. However, the transaction is processed by a large number of nodes in the blockchain system, which causes all nodes to pay attention to whether the operations and data related to the transaction are open to all nodes. This transaction process will undoubtedly lead to the risk of privacy leakage, especially in some data-sensitive scenarios such as voting schemes and the medical industry.

Therefore, it is extremely important to properly handle the relationship between sensitive private data and smart contracts. For example, when using smart contracts to implement an auction system based on the Ethereum system, an auction system with a closed auction mechanism should be created. Its sealed bidding rules are as follows: In the bidding stage, bidders submit sealed bids to the auctioneer, and the amount of each bidder is not visible to others. If a bidder bids again, the bid will not be sealed and the winning bidder will be selected by comparing the bid amounts. Thus, in this special scenario, the most important private data such as transaction amounts, have no privacy at all, which seriously damages the closed auction mechanism.

2) CONTRACT DATA PRIVACY ISSUES

Smart contracts must be distributed to blockchain nodes before execution. However, the physical hardware configuration of blockchain nodes varies significantly. Some nodes exhibit poor performance and have potential security risks [23]. If the smart contract code is not written correctly, these nodes with poor security are prone to loopholes in the process of executing the contract code [24], resulting in the leakage of contract data privacy.

For example, by illegally entering a weak node in the blockchain network, an attacker can steal all smart contracts deployed on that node. For some application scenarios that are sensitive to contract privacy, the leakage of smart contracts may be fatal. The attacker will deduce the contract associated with the contract through the acquired smart contract, analyze the actual behavior of the user of the relevant contract, and compare it with real-life behavior. The user's identity can be obtained from the user's real identity, and the final result is that all of the user's information will be made public. Alternatively, an attacker can use the acquired smart contract to deduce potential problems with the contract itself, and then use hacking to attack the entire system and steal sensitive data, which may also lead to catastrophic security issues such as loss of confidential information.

The privacy issue of contract data may lead to de-anonymizing attacks on the blockchain or smart contracts by attackers. Therefore, it is imperative to seek solutions to the privacy and security issues of smart contracts.

C. CONTRACT SECURITY VULNERABILITIES ARE VULNERABLE TO ATTACK

The emergence of smart contracts has enriched blockchain application scenarios, turning blockchain platforms into powerful decentralized systems. However, smart contracts are more vulnerable to attacks than normal programs. This is because, (1) because smart contracts are used to manage digital currency on the blockchain platform, attackers have strong willingness and motivation to attack smart contracts to illegally acquire and possess digital currency assets; (2) the execution of smart contracts in the blockchain system can theoretically guarantee the trustworthiness of contracts. However, in reality, there may be potential security loopholes in smart contracts, and loopholes will make the execution results of contracts unpredictable, making contracts unequal, and thus losing the meaning of the smart contract.

Unlike traditional contracts, which are described in natural language, smart contracts are generated, verified, and executed through program code. After the smart contract is written in a high-level language, the contract is compiled into bytecode, and then executed by transaction events, which encounter various security threats. Potential security vulnerabilities in smart contracts mainly come from three levels: the high-level language for writing smart contracts, the virtual machine that supports the operation of smart contracts, and the blockchain system in which the smart contracts are located. At the high-level language level, common security vulnerabilities include integer overflow, unchecked return values, arbitrary address writing, denial of service, and uninitialized variables etc. Smart contracts are decentralized applications running on a blockchain, and the security threats they face are closely related to the running environment. At the virtual machine level, common security vulnerabilities include code injection, short address attack, and timestamp dependence, while at the blockchain level, common security vulnerabilities are conditional competition and lack of randomness. Ethereum is the earliest open-source smart contract blockchain platform. This study considers the Ethereum smart contract system as an example to discuss and analyze the security vulnerabilities of smart contracts. Developers

TABLE 2. Smart contracts execute efficiently application.

Refence	Year	Smart Contract	Application feature
Dickenson et al.	2017	Ethereum	This is a smart contract execution framework that supports a multicore architecture, allowing miners and validators to execute independent, conflict-free smart contracts in parallel
Das et al.	2018	Ethereum	YODA is a solution for implementing computation-intensive smart contracts (CIC). YODA executes CIC chain by selecting one or more Execution sets (ES), By using the multi-round Adaptive Consensus using Likelihood Estimation (MIRACLE) algorithm based on sequential hypothesis testing, YODA can execute CIC efficiently and correctly with high probability.
Karl et al.	2020	Ethereum.	ACE is a smart contract system that supports asynchronous concurrent execution and is designed to execute smart contracts with higher computing, storage and communication costs on the blockchain.
Wellington et al.	2020		A cryptocurrency Iota Tangle is proposed. Iota Tangle is also a distributed communication protocol based on consistency algorithm.
CITA team	2020		CITA is a smart contract-enabled blockchain microservice architecture for enterprise applications, aiming to provide a stable, efficient, flexible and future-ready platform for enterprise blockchain applications.
Sariboz et al.	2021	Ethereum	A framework for off-chain execution and validation of CIC that does not require a trusted execution environment, supports computations without deterministic results, and supports general-purpose computations written in high-level languages,
Zhang et al.	2021	Mictract	A framework for smart contract microservitization is proposed. Then, Mictract, a smart contract platform based on microservices, was designed to deploy and monitor smart contracts.
Fang et al.	2021	BFT-SMaRt	A new two-phase framework based on trusted hardware Intel SGX is proposed, which can avoid the re-execution of all smart contracts on all nodes and improve the parallelism between nodes.
Chen et al.	2022	Bitcoin	A multi-payment channel (MPC) network is proposed to support multiple payments using the same intermediate channel at the same time, thus greatly expanding the application scenarios of payment channels.
Liu et al.	2022	Ethereum	A new smart contract execution paradigm is proposed. This is a new paradigm for parallel and asynchronous smart contract execution that requires neither extensive coordination nor (adversarial) live locks, and it requires a small group size.

use the scripting language of the Ethereum system itself to write smart contracts, and the related technologies are still immature, so it is inevitable that there will be loopholes and thus are vulnerable to attack.

Because the smart contracts deployed on the blockchain are irreversible, once the potential security problems of high-level language tools, virtual machine environments, and blockchain platforms occur, they are difficult to repair, and the resulting economic losses will be irreversible [25]. Simultaneously, the anonymity of the blockchain may provide convenience for malicious users, which in turn leads to security issues in real-world applications. In 2017, a vulnerability was discovered in the multi-signature wallet of Parity Wallet, which made the wallet unprotected, and hackers could easily invade the wallet to gain ownership, finally leading to the theft of three large-value currency accounts in the wallet.

An increasing number of incidents of hackers attacking smart contracts show that the security problems of smart contracts are serious. The security vulnerabilities of smart contracts, as well as research on security vulnerabilities of contracts, should receive more attention.

IV. PROGRESS ON KEY ISSUES OF SMART CONTRACTS

A. PROGRESS ON THE PROBLEM OF LOW CONTRACT EXECUTION EFFICIENCY

The low efficiency of smart contract execution significantly restricts its application and promotion. Extensive research has been conducted on related issues, as listed in Table 2.

In the smart contract blockchain system, before all smart contracts are put on the chain, they are executed serially by the miner nodes, and then by the contract validator again serially. Serial execution cannot make full use of the advantages of the concurrent multicore cluster architecture, which severely restricts system throughput and leads to low contract execution efficiency. Dickenson *et al.* [26], in response to this problem, proposed a smart contract execution framework that supports a multi-core architecture, allowing miners and validators to execute independent, conflict-free smart contracts in parallel, with a speedup ratio of 1.33 for miners and 1.69 for validators, respectively. Thus, the efficiency of smart contract execution is significantly improved. Karl *et al.* [27] proposed a system that supports Asynchronous and Concurrent Execution of complex smart contracts (ACE), which aims to execute smart contracts with higher computational, storage and communication costs on the blockchain. The ACE system is based on the off-chain execution model. The contract issuer designates a group of service providers to execute the contract code independent of the consensus layer. It supports more complex smart contracts than the Ethereum standard. With an off-chain execution, the contract is more efficient and flexible. The system concurrency control protocol allows contracts to call each other across the boundaries of service providers, but does not require all service providers to trust each other, which allows the system to process multiple signatures concurrently and reduces the time cost of verification. Experiments show that even if the contract has an average of 200 state changes per transaction, the total verification cost of the block is very small at only 0.12 seconds per block. When multiple verification signatures are performed in parallel, the signature of each service provider reduces the

total block verification time by 0.1 milliseconds. Therefore, it can be seen that the ACE system reduces the time required for the signing and verification process in the execution of complex contracts, making contract execution more efficient. Silvano and Marcelino [28] proposed cryptocurrency, the Iota Tangle. The authors pointed out that the Iota tangle is also a distributed communication protocol based on a consensus algorithm. This protocol requires validating and referencing two existing transactions in the ledger to create a new transaction. Based on proof-of-work, each new transaction affects the total weight, which is sufficiently high to be legal. For old transactions to be confirmed in time, a coordinator is introduced and checkpoints are issued to periodically confirm transactions. The architecture of the protocol can create a barrier-free distributed network without miners, so that contract transactions can be executed in parallel at a lower computational cost, thereby effectively improving the efficiency of contract execution.

Owing to the complexity of Compute-intensive Contracts (CIC) leads to inefficient execution on the blockchain and high operating costs per CIC. In response to this problem, Das *et al.* [29] proposed a solution called YODA to implement CIC. YODA executes the CIC off-chain by selecting one or more execution sets (ES), and innovatively uses MultI-Round Adaptive Consensus using Likelihood Estimation (MIRACLE) based on sequential hypothesis testing to perform CIC efficiently and correctly with high probability. The experimental results show that using the YODA scheme expands the maximum transaction value of Ethereum by a factor of 450, and that as the number of simultaneous dense transactions increases, the average CIC execution efficiency on each node also increases. Sariboz *et al.* [30] proposed a framework for the off-chain execution and verification of CIC. The framework does not require a trusted execution environment, supports computations without deterministic results, supports general-purpose computations written in high-level languages, outsources CIC computations to third parties, and computes and generates correctness proofs for CIC. This verification can be completed in polynomial time. Using this method, the client can verify and return the correct result more efficiently, and make contract execution more efficient.

For the problems of low deployment efficiency in the development, operation, and maintenance of smart contracts, as a highly autonomous, distributed, and decentralized application, microservices have similar characteristics to smart contracts: therefore, these problems can be solved by smart contract microservices. For example, in 2020, the Cryptape Inter-enterprise Trust Automation (CITA) technology underlying technology development team, based on the CITA-Cloud framework, proposed a blockchain microservice architecture CITA that supports smart contracts for enterprise-level applications, which is designed to provide a stable, efficient, flexible and future-proof operating platform for enterprise-level blockchain applications, and released The ''CITA Technical White Paper'' [31]. The architecture separates executor from the chain. An executor is only responsible for computing and executing transactions, whereas a chain is responsible for storing transactions. The separation of calculation and storage greatly improves the transaction processing capacity and execution efficiency of transaction contracts. Zhang *et al.* [32] proposed a microservice framework for smart contracts, and designed a microservice-based smart contract platform called Mictract to complete the deployment and monitoring of smart contracts. The core advantage of Mictract is that it has the basic functions of the Blockchain as a Service (BaaS) platform. At the same time, it can provide cloud-based third-party services for people or organizations building blockchain applications, allowing users to build, host, run, and monitor their own blockchain applications on the cloud platform, forming a tool chain that supports the development, operation, and maintenance of smart contracts. The experimental results show that Mictrac reduces the average time of deploying contracts by 225.6s compared with the script method. Mictrac is satisfactory in terms of smart contract deployment, upgrade efficiency and smart contract operation monitoring. There is a set of design principles between microservices and smart contracts, as listed in Table 3.

Due to the lack of mutual trust among nodes, for a batch of smart contracts contained in a block, the traditional two-stage smart contract concurrency can only realize the concurrency within a single node, but cannot realize the parallel execution of contracts between nodes. To solve this problem, Fang *et al.* [33] proposed a new two-phase framework based on trusted hardware Intel SGX, which can avoid the re-execution of all smart contracts on all nodes and improve the parallelism between nodes. And consistency between nodes is achieved directly through state replication rather than re-executing the transaction. We design a pre-execution mechanism for untrusted smart contracts in memory to obtain all the state data that the smart contracts need to access in batches, so as to reduce frequent enclave transformations during the execution of smart contracts. In addition, a method for generating compact read and write sets and a data structure named Merkle Forest are proposed, which can generate compact Merkle multiple proofs for initial data in untrusted memory in parallel and can quickly verify the correctness of data passing through enclave. Finally, all the techniques proposed in this paper are integrated into the open source system BFT-Smart to evaluate the approach of the framework in a distributed environment. Experimental results show the effectiveness of the proposed method.

Blockchain-based cryptocurrencies are severely limited in terms of transaction throughput and latency. One promising solution to this problem is payment channels, which allow untrusted payments between two peers without draining the blockchain's resources. The linked Payment Channel network (PCN) enables payments between two peers through a series of intermediate nodes that forward and collect payment fees. However, most existing proposals only use the shortest path as a path for transactions, which results in frequently reused

TABLE 3. Design principles for microservices and smart contracts.

Design principles	Microservice	Applicable to smart contracts	Reason
High cohesion Low coupling	Small service volume Single responsibility Lightweight communication	☑	Smart contracts have specific scope and responsibilities
High degree of autonomy	Independent development Independent deployment Independent publishing Process isolation	☑	Smart contracts run independently in the sandbox environment
Distributed Decentralized	Distributed system architecture A decentralized database	☑	Smart contracts are deployed on distributed networks
Centered on the business	Microservices represent business, respond to business changes quickly, organize the team according to business, and the team is only responsible for business	☒	Smart contracts cannot quickly respond to business changes, so the team needs to take into account smart contract development and operation as well as blockchain network maintenance

channels being quickly exhausted. In addition, most existing PCNS are designed almost exclusively for payments between two parties, resulting in limited application scenarios. When multiple payments use the same middle channel, two PCNS cannot realize simultaneous payment. Chen *et al.* [34] proposed a multi-payment channel (MPC) network, a payment channel proposal that supports multiple payments using the same intermediate channel at the same time, thus greatly expanding the application scenarios of payment channels. It allows any number of users to participate in the same multi-party channel and conduct atomic multi-party transactions based on the original TPC. In order to support as many transactions as possible through the blockchain network and avoid repeated channel updates, we propose a channel selection strategy to select high-quality transaction channels so as to improve the success rate of transactions and transaction throughput.

Blockchains are plagued by low throughput and high latency, which hinder their widespread adoption of more complex applications such as smart contracts. Liu *et al.* [35] proposed a new paradigm of smart contract execution. This is a new paradigm for parallel and asynchronous smart contract execution that requires neither extensive coordination nor (adversarial) live locks, and it requires a small group size. The authors suggest two ways to put this paradigm into practice. First, the new paradigm is applied to Ethereum, and it can enable Ethereum to support parallel and asynchronous execution without any hard bifurcation. Then, a new public and permissive blockchain SaberLedger based on the new paradigm is proposed, and its performance is demonstrated by implementing a prototype. It can effectively improve the throughput of transactions and protect the security of transactions.

To solve the problem of low efficiency in contract execution, the technologies such as concurrent execution, off-chain computing, and contract microservices can be adopted. If the concurrent execution method is adopted, the smart contract execution framework of the multicore architecture may face the compatibility problem of "soft fork" in the future, as well as the compatibility problem with the smart contract system. Using off-chain computing or executing computationally intensive functional contracts by a third-party computing platform will lead to some contract correctness verification and contract security issues. Systematic research methods for the microservice method of smart contracts to support the evaluation of the microservice framework of smart contracts are still lacking. Additionally, in terms of data scalability, it is necessary to increase the data pluggability for blockchain networks.

B. PROGRESS OF CONTRACT DATA STORAGE EXPANSION PROBLEM

The difficulty of data storage expansion is not unique to blockchain smart contracts, and traditional database systems have similar problems, as listed in Table 4.The idea of sharding comes from the expansion technology of the database, which improves overall performance by dividing the database into multiple small and processable parts. Sharding technology is also effective when blockchain contract data are difficult to expand. Luu *et al.* [36] designed a secure sharding protocol called *Elastico* for open blockchain. Technically, Elastico can increase the throughput of contract transactions linearly with an increase in network computing power, and can tolerate illegal nodes with a maximum of one-quarter computing power. Experiments show that when the network size is 400 and 800 nodes, the delays to reach consensus are 103 and 110 s, respectively, and the consensus time does not change significantly, confirming that Elastico is highly scalable and extremely usable in the next generation of cryptocurrencies. Gencer *et al.* [37] proposed Aspen, a service-oriented blockchain-sharding protocol. Aspen enables the secure scaling of blocks as the number of services increases, protecting the blockchain from block forks. Aspen can be instantiated according to the characteristics of different blockchain systems, which reduces the resource requirements and shortens the boot time of the

TABLE 4. Progress of contract data storage expansion application.

Refence	Year	Smart Contract	Application feature
Luu et al.	2016	Bitcoin	An open block chain security sharding protocol Elastico is designed. Technically, Elastico's network throughput increases as the network's computing power increases, and it can tolerate up to a quarter of the computing power of illegal nodes.
Gencer et al.	2016	Bitcoin	Aspen, a service-oriented block chain sharding protocol, is proposed. Aspen allows blocks to scale securely as the number of services increases, protecting the blockchain from the impact of fork.
Wen et al.	2018	Zilliqa	A sharding technology for blockchain systems has been designed that includes Zilliqa, a blockchain-based sharing protocol,
Duong et al.	2018	Bitcoin	A multi-mode smart contract system PRUNE is proposed, which supports full mode and pruning mode.
zhu et al.	2019	Hyperchain	Based on the research of key technologies of alliance chain, the architecture design and implementation scheme of high performance alliance blockchain are proposed.
Taxa Website platform	2020		The developers proposed Taxa, a blockchain that addresses the difficulty of scaling contract data by using reliable hardware to create a separate external execution environment for smart contracts,
Wang et al.	2020		An extended smart contract system is proposed. The system integrates Oracle mechanisms into the input and output modules of smart contracts.
Kim et al.	2020		A selective compression scheme for lightweight nodes in blockchain system is proposed to prevent the accumulation of compression results.
Wang et al.	2021	Consortim Blockchain	Based on the static and dynamic access control methods of smart contracts, stored information in the form of dual blockchains, and stored the user's attribute information in the attribute blockchain maintained by the endorsement node

system. Therefore, Aspen offers a new method to improve the scalability of blockchains. Wen *et al.* [38] developed a sharding technology for blockchain systems, which designed a blockchain-based sharing protocol, *Zilliqa*. *Zilliqa* can process about 1,400 transactions per second on thetest network, and the transaction efficiency is significantly improved compared to the *Ethereum* blockchain, but for open large-scale smart contract projects, there are still problems of inefficiency and inability to handle large-scale transactions.

In response to the continuous expansion of contract transaction data, which leads to the problem that the node data download and synchronization time is too long, Duong *et al.* [39] proposed a multi-mode smart contract system *PRUNE*, which supports full mode and pruning mode. Based on the two operation modes, the full mode and pruning mode are formed, which record the complete and compressed contract transaction data respectively. When the system is in the pruning mode, unnecessary information can be deleted from the ledger, freeing up the storage space, forming lightweight nodes, and solving the problem of blockchain contract data being difficult to backup and expand. Zhu *et al.* [40] proposed an architectural design and implementation plan for a high-performance consortium blockchain based on the research foundation of the key technologies of the consortium chain, combined with the existing securities trading system functions of the Executive Committee. This solution realizes a high degree of combination of business logic and consensus separation, storage optimization, digital signature verification optimization, and other optimization strategies to improve the performance of the consortium chain, and build a high-performance consortium chain.

In 2020, developers of the Taxa Website platform proposed a blockchain called *Taxa* [41]. The Taxa uses reliable hardware to create a separate external execution environment for smart contracts to address the problem of contract data being difficult to expand. The public chain acts as a ''consensus layer'' and records the final token payment and contract state transition results. The Taxa platform isolates the public chain consensus and the execution of smart contracts, puts specific on-chain operations under chain control, and runs efficient, safe, and reliable smart contracts on the chain.

Wang *et al.* [42], based on the static and dynamic access control methods of smart contracts, stored information in the form of dual blockchains, and stored the user's attribute information in the attribute blockchain maintained by the endorsement node. With an increase in the number of requests, the average request time is stable within 12–13 *ms*, and the request time will not increase because of high concurrent requests. This data storage and access control method can effectively solve the problems of data security and system scalability.

It is impossible to verify the authenticity and integrity of the smart contract triggered by the off-chain data. This poses a challenge to the execution results of blockchain smart contracts. If the accuracy of the acquired off-chain data is not guaranteed, then the smart contract cannot guarantee the correctness of its execution results. Wang *et al.* [43] proposed an extended smart contract system using Oracle Machine. The system integrates Oracle mechanisms into the input and output modules of smart contracts. As the interface between blockchain and the off-chain world, Oracle mechanism can better realize the interaction between smart contracts and off-chain data, and improve the integrity of smart contracts in the execution process. By designing the incentive Mechanism of Oracle Mechanism for nodes, nodes can be encouraged to treat their input data more carefully. There are also rewards for nodes that provide accurate, realistic data. As a result, the efficiency and accuracy of the whole system have been greatly improved.

TABLE 5. Improve intelligence and privacy application.

Refence	Year	Smart Contract	Application feature
Kosba et al.	2016	Bitcoin，Ethereum	Hawk is a smart contract system that ensures confidentiality by enforcing contracts down the chain and issuing only zero-knowledge proofs on the chain.
Hunt et al.	2016		Ryoan is a distributed sandbox platform that uses SGX to limit the privacy of sensitive data.
Zhang et al	2016	Ethereum	Town Crier is a trusted data entry system that allows users to send private data requests,
Microsoft coco framework	2018	Ethereum,Quorum, Corda	A parallel and independent smart contract system for migrating existing smart contract systems (such as Ethereum) to the trusted execution environment provided by software guard extensions (sgx).
Cheng et al.	2018	Ethereum.	Ekiden is a system that addresses important privacy issues by combining blockchain with a strong law enforcement environment. Ekiden implements efficient, confidential and highly scalable smart contracts based on a novel architecture that separates consensus from execution.
Kalodner et al.	2018		Arbitrum is a cryptocurrency system that supports smart contracts, which has better privacy compared to previous solutions.
ORIGO	2019	Ethereum	Origo is a project designed to strengthen the privacy protection of Ethereum smart contracts, using a password promise to protect the confidentiality of contract records, combined with reliable hardware and zero-knowledge proof to guarantee the confidentiality of contract processes and results.
Kopp et al.	2019		Proposed a decentralized storage system called PriCloud.
Kapsoulis et al.	2020	Alastria Network	This paper proposes a decentralized privacy protection approach, which uses two modes to develop smart contracts to protect users' privacy on the enterprise blockchain.
Hao et al.	2021	Ethereum	This is a smart contract-based access control framework that enables owners to implement resource access control in a reliable, auditable, and scalable manner.
Wan et al.	2022	Ethereum,	Zk-dasnark is an efficient authentication zero-knowledge proof protocol that enables privacy and authenticity of off-chain data input for smart contracts on blockchain.

For the problem that blockchain nodes need huge storage to cope with the increasing size of the blockchain ledger over time, several compression schemes have been proposed to alleviate this storage problem by compressing the blockchain ledger based on redundancy, modular functions, and hash functions. This requirement leads to conditional participation and verification of participants, thus weakening the decentralization of blockchain systems. However, these schemes have the limitation of accumulating compression results to validate reserved blocks. Accumulation gradually reduces the storage capacity of blockchain ledger within the storage capacity of resource-limited nodes, thus reducing the verification capability of nodes. Kim *et al.* [44] proposed a selective compression scheme of lightweight nodes in blockchain system to prevent the accumulation of compression results. The checkpoint chain is the second blockchain that stores checkpoints that compress existing blocks through the block Merkle tree. An update process is also proposed to prevent the accumulation of checkpoints by combining them. Because a large number of blocks can be verified with only a few checkpoint updates, resource-limited blockchain nodes can reduce the storage of blockchain ledgers and achieve high verification capabilities. Finally, compared with the existing compression schemes, the proposed scheme can achieve an average reduction in storage overhead and an average improvement in verification capability, which are 76.02% and 13.90%, respectively. In addition, when performing the update process, the corresponding performance improvements were 86.14% and 15.44%, respectively.

To address the problem of smart contract data being difficult to expand, technologies such as sharding protocols, side chains, and lightweight nodes can be used to improve the scalability of the blockchain. Sharding protocols and side chains are mainly used to improve the overall scalability of contract data. Lightweight nodes are mainly to improve the scalable performance of a single node. However, these technologies still face challenges in terms of solving the scalability problem of contract data. For example, the application of these technologies may lead to increased computing power and node centralization. Owing to the extremely low computing power of side chains, it is difficult to ensure the security of transactions and blocks.

C. PROGRESS OF SMART CONTRACT PRIVACY ISSUES

Smart contracts are a new stage in the development of blockchain technology, and their privacy protection strategies and methods have changed. The privacy protection of smart contracts typically adopts a combination of zero-knowledge proof and a trusted execution environment, as listed in Table 5.

Kosba *et al.* [45] proposed a smart contract system, *Hawk,* which ensures confidentiality by executing off-chain contracts and publishing only on-chain zero-knowledge proofs. In addition, the hawk is designed as a single node, and its availability is not high. *Hawk* supports only limited contract types and does not have the general functionality of other similar systems. The idea of combining a ledger and trusted hardware to execute smart contracts is briefly mentioned in *Hawk*, but omits key system design issues: for example, its permissionless ''proof of publication'' ignores the lack of trust in some environments and technical difficulties caused by the wall clock time. Zhang *et al.* [46] proposed a trusted

data input system called Town Crier. Town Crier allows users to send private data requests. Specifically, the contract encrypts the request with the public key of the town crier before sending the request. Decrypt with a private key to ensure that other users in the blockchain cannot view the requested content. The Microsoft coco framework [47] is a parallel and independent smart contract system for migrating existing smart contract systems (such as Ethereum) to the trusted execution environment provided by software guard extensions (sgx). However, if we look at the information, only one white paper provides a general overview, and details of the protocol and implementation have not been released.

Smart contract privacy protection systems are now based on trusted hardware, particularly Intel's sgx, which is used in a variety of applications in distributed systems. Hunt *et al.* [48] Ryoan, a distributed sandbox platform, used sgx to limit privacy leakage of sensitive data. The system cannot address the state integrity and confidentiality issues throughout its life cycle. Cheng *et al.* [49] proposed Ekiden, a system that addresses important privacy concerns by combining a blockchain with a robust law enforcement environment. Ekiden implemented efficient, confidential, and highly scalable smart contracts based on a novel architecture that separates consensus and execution. Ekiden supports secure communication between long-term smart contracts across different trust domains, can implement mitigations to maintain data integrity, and limits data leakage to mitigate potential failures in trusted runtime environments. Therefore, Ekiden allows independent services to run longer than a single node, user, or project and ensures the privacy of smart contracts.

Kalodner *et al.* [50] proposed a cryptocurrency system supporting a smart contract *Arbitrum* with better privacy than previous solutions. *Arbitrum* used a customized virtual machine architecture to reduce the cost of on-chain dispute resolution, taking most of the implementation of virtual machine behavior off-chain and reducing the cost of on-chain resolution. The internal state of the virtual machine is not revealed to the validator unless there is a dispute between the two sides of the transaction. Even in the event of a dispute, transacting parties can only obtain information about one step of execution, and the vast majority of states are opaque to transacting parties. This unique system mechanism guarantees the flexibility and privacy of contract data. Origo, a project aimed at strengthening the protection of the privacy of Ethereum smart contracts, was proposed [51]. It utilizes cryptographic promises to protect the confidentiality of contract records, combined with reliable hardware and zero-knowledge proofs to guarantee the confidentiality of contract processes and results. In addition, the contract requires certain currencies to be pledged to prevent malicious acts by both parties or to reveal the privacy of others. In the event of malicious activity, the pledged currency is confiscated. Unlike traditional blockchains, not all origo nodes fulfil their contracts. Instead, participants chose a node on the network responsible for fulfilling the contract and generating zero knowledge.

Centralized providers have a negative impact on the privacy of their users because they are able to read and collect all kinds of data about their users and even pay to associate it with their identity. On the other hand, decentralized storage solutions like GNUnet lack vendor involvement because there is no viable business model. Kopp *et al.* [52] proposed a decentralized storage system called PriCloud to address these issues. It allows users to pay storage providers in anonymous currency. The blockchain, where transactions are encrypted and hardened, makes payments untraceable and unlinkable. Storage smart contracts that determine the duration of storage are included in the blockchain, allowing payments to be executed automatically. The storage provider may make payments from the contract if it provides valid proof of storage of the documents specified in the contract at certain points in time. Therefore, the storage provider will only receive payment if it stores the user's files in good faith. Our PriCloud system provides incentives to participate by offering privacy-protecting financial incentives for participating as a storage provider.

In response to the privacy issue of smart contract transactions for enterprise blockchain users, in the context of the standardization of Know Your Customer (Kyc), Kapsoulis *et al.* [53] proposed a decentralized privacy protection method that was developed using two-modes smart contracts to realize user privacy protection on the enterprise blockchain. Through the public Kyc smart contract, the user registration and off-chain storage of Kyc information are realized. The repository information is managed through a private-kyc smart contract. Therefore, sensitive information is not stored in the blockchain. In this way, users' sensitive information cannot be accessed from blockchain nodes, which protects the privacy of blockchain smart contracts.

The traditional access control model mainly relies on a centralized trusted server to mediate every attempt of clients to access resources, which is faced with serious challenges of single point of failure and lack of transparency. Hao *et al.* [54] proposed an access control framework based on smart contracts, which enables owners to implement resource access control in a reliable, auditable, and scalable manner. Access control contracts are deployed on the blockchain to flexibly manage attribute-based resource access policies and make trusted access decisions for customers. A set of properties is distributed to clients through an off-chain signature signed by the owner to determine their permissions without consuming expensive on-chain storage space. Finally, we implemented an experimental prototype on the Ethereum test network and conducted extensive experimental and theoretical analysis to evaluate its scalability and efficiency.

When blockchain technology is applied to decentralized traditional applications, blockchain verifiers may need to obtain sensitive off-chain data to execute smart contracts. Wan *et al.* [55] proposed an efficient authentication zero-knowledge proof protocol called ZK-DASnark.

ZK-AuthFeed, an off-chain data feed scheme with zero knowledge authentication, is designed. It combines ZK-SNARK and digital signature to achieve privacy and authenticity of off-chain data input of smart contracts on blockchain. Following the strategy of ''compute off the chain, verify on the chain'', zkAuthFeed can significantly reduce the computing costs of blockchain verifiers.

For the privacy protection protocols of various smart contracts, a trusted execution environment is generally used as a trusted computing environment for smart contract execution, and the zero-knowledge proof method is usually used to provide proof conditions for the execution of smart contracts. However, these protocols are computationally expensive, difficult to implement, and depend on hardware and cryptography to ensure state integrity and confidentiality.

D. PROGRESS OF SMART CONTRACT SECURITY ISSUES

Smart contracts involve the transaction of digital assets and are open source, so they are inherently vulnerable to attack and destruction. Once potential security loopholes are exploited and attacked, the security of the entire blockchain system is seriously threatened, which is bound to cause incalculable losses. Usually, before a smart contract is executed, it needs to use detection tools are required to eliminate the existence of security loopholes, to ensure the credibility and reliability of its execution in the chain. Detection tools often use automated vulnerability mining techniques that can efficiently identify potential security vulnerabilities in smart contracts.

Automated vulnerability mining is an important research field in software vulnerability mining. The primary methods used include fuzzing testing, symbolic execution, formal verification and other technologies. As a growing class of decentralized applications, smart contracts are very different from traditional applications in terms of the operating environment, life cycle, application characteristics, and traditional vulnerability mining techniques are difficult to use directly for mining the potential vulnerabilities of smart contracts. Existing research focuses on the application of traditional software vulnerability mining techniques to smart contracts, as listed in Table 6.

1) FUZZING

Fuzz testing is a relatively effective software-analysis method. The main idea is to build a large number of test cases for programs, track abnormal behaviors during program execution, and identify and loopholes in programs [56]. Building test cases is the key step in fuzzing. Generally, there are two test case construction methods: generation and mutation. Generation-based fuzzy testing is suitable for situations where the input format of the test is relatively clear or the format requirements are relatively strict, such as programs that interact with file management systems or protocols. The tester generates test cases according to the input format to improve efficiency. Mutation-based fuzzing tools change the seed based on the original input and feedback from the program. The mutation techniques include bit flipping, incrementing, decrementing, and copying.

Fuzzing technology is widely used in traditional program vulnerability mining, and has been proven to be very effective. The American Fuzzy Lop (AFL) [57] is a very popular fuzzing tool, which is mainly oriented to the vulnerability mining of C/C++ applications on the Linux platform. The tool adopts a fuzzing scheme based on coverage feedback to guide the seed mutations. In applications, at least hundreds of memory corruption vulnerabilities in popular applications, public function libraries and operating system kernels have been discovered. On this basis, a large number of follow-up studies have focused on optimizing the seed selection strategy, mutation strategy, path recording strategy, etc. [58]–[61] to improve the efficiency of vulnerability mining, and have achieved very good experimental results. These findings demonstrate that fuzzing is a very effective method for exploiting vulnerabilities.

Liu *et al.* [62] designed a *ReGuard* tool based on a fuzzing method to identify reentrance errors in smart contracts. This tool introduces a finite perpetual motion machine-reentrant perpetual motion machine in the core detector. By modeling and then calling the reentrancy function to observe whether the ether transmission phenomenon occurs, and at the same time, a check report is generated, and the user checks the report to confirm whether the reentrancy occurs. Liao *et al.* [63] proposed a vulnerability detection scheme based on the combination of machine learning and fuzzing, which can learn to detect unknown vulnerabilities, improve the vulnerability detection rate, and affect the detection of integer overflow vulnerabilities.

In 2018, the security research organization Trail of Bits released Echidna [64], an open-source smart-contract fuzzing solution, on its blog. Echidna provides a relatively complete Ethereum smart contract fuzzing framework that can analyze and simulate the execution of smart contract source code, and generate random transaction data that conform to the contract calling specification to test the contract fuzzily. Echidna introduces coverage information to detect the execution efficiency of fuzzing, but does not delve into more effective seed generation strategies. In addition, Echidna does not provide a general vulnerability detection method, and testers still need to add specific vulnerability detection code to the contract source code, and judge whether there is a vulnerability through the return value status. Although Echidna is a relatively perfect industrialization-fuzzing testing framework for smart contracts, it still fails to solve the two major challenges in fuzzing testing of smart contracts: it is difficult to generate effective test cases and to effectively detect vulnerabilities.

Jiang *et al.* [65] proposed ContractFuzzer, a smart contract fuzzing research scheme that is mainly used for the vulnerability detection of smart contracts. ContractFuzzer was developed based on Geth, the client of the Ethereum Go language version, and performs offline vulnerability detection by recording the command log when the smart contract is executed. Detection models have been established

TABLE 6. Automated vulnerability mining application.

Method	Refence	Year	Smart Contract	Application feature
Fuzz testing	Liu et al.	2018	Ethereum	Reguard is a tool based on a fuzzy testing approach that looks for reentrant errors in smart contracts.
	Liao et al	2019	Ethereum	A vulnerability detection scheme based on machine learning and fuzzy testing is proposed.
	Jiang et al.	2018	Ethereum	This is a relatively perfect fuzzy test framework for Ethereum smart contract, which can analyze and simulate the execution of the source code of smart contract, and generate random transaction data in accordance with the contract invocation specification, so as to carry out fuzzy test on the contract.
	Trail of Bits released	2018	Ethereum	ContractFuzzer is a smart contract fuzzy testing research solution, mainly used for smart contract vulnerability detection.
	Zhang et al.	2020	Ethereum	ETHPLOIT is a smart contract fuzzer that addresses complex and unsolvable constraints in smart contract execution using symbols and blockchain attribute vulnerabilities related to timestamps and block numbers.
	Ashraf et al.	2020	Ethereum	GasFuzzer is a new technology that consists of two strategies.
	Zhou et al.	2021	Ethereum	The SMARTGIFT method learns from the transaction records of real-world smart contracts in order to generate practical test inputs for the new smart contracts being tested.
	Choi et al.	2021	,	SMARTIAN is a method for enhancing fuzzy testing of smart contracts through static and dynamic data flow analysis.
Symbolic execution	Luu et al.	2016	Ethereum	Oyente is an automated contract vulnerability mining system. Oyente provides a simplified contract symbol execution engine for developers to write contracts easily.
	Consensys company	2017	Ethereum	Mythril is a symbolic execution engine that analyzes all kinds of EVM bytecode based contracts.
	He et al	2019	Ethereum	ILF (Imitation Learning Based Fuzzer) is a fuzzy testing scheme for smart contracts based on symbolic execution and neural network. It is committed to generating better test cases and transaction sequences in fuzzy testing of smart contracts.
	Mossberg et al.	2019	Ethereum	Manticore is a smart contract dynamic symbolic execution engine that improves analysis efficiency and code coverage by externalizing variable values.
	Wang et al.	2020	Ethereum	ContractWard, which is a model to detect six types of vulnerabilities of smart contracts effectively and efficiently based on extracted static features.
	Chen et al.	2021	Ethereum	DefectChecker is a symbol-based execution method and tool that can be used to detect eight contract defects that can cause bad behavior in smart contracts on the Ethereum blockchain platform.
	Jin et al.	2022	Ethereum, EOS	EXGEN is a cross-platform open source framework for automatically generating exploits for smart contracts on EOS and Ethereum.
	Hwang et al.	2022		CodeNet is a new code-targeted CNN architecture. To improve the performance of CodeNet, a data preprocessor was designed to convert smart contracts into images while maintaining locality.
Formal verification	Hirai et al.	2017	Ethereum	Formalized description of instruction semantics in ethereum virtual machine, based on description results, manually prove the security of a program.
	He et al.	2018		SPESC is a smart contract protocol language, which defines some protocols of smart contracts, supports cooperative design of contracts, and provides technical support for multiple people to write smart contracts.
	Kalra et al.	2018	Ethereum	ZEUS is an automated formal validation tool that supports five security vulnerabilities in smart contracts.
	Park et al.	2020	Ethereum	The K framework is used to formally verify the smart contract with deposit function on Ethereum.
	Antonino et al.	2020	Ethereum	Solidity language has been abstracted based on how ethereum smart contracts behave.
	Yang et al.	2020	Ethereum	A symbolic Process Virtual machine (FSPVM-E) is designed by combining symbolic execution with higher-order logic theorem proving.
	Nam et al.	2022	Ethereum	This is a novel form verification technique, using ATL (Alternating-) Time Temporal Logic) model checks to analyze blockchain smart contracts.
	Park et al.	2022		A Dutch auction trading system based on smart contracts was modeled and validated using the UPPAAL model checking tool for time attributes, and time constraints were set for each template and state during the modeling process.

for seven types of vulnerabilities, including gas deficiency, exception delivery, reentrancy, timestamp dependency, transaction sequence dependency, code injection, and asset freezing, and use the instruction log to detect whether the current contract execution triggers the vulnerability. In terms of input generation, ContractFuzzer generates transactions randomly by randomly generating call parameters, transaction amounts, and transaction-sending addresses. In addition, to trigger reentrancy vulnerabilities, it specially designs a unique attack proxy contract, and attempts to trigger the reentrancy vulnerability in the tested contract by calling the tested contract through the attack proxy contract.

Zhang *et al.* [66] designed a smart contract fuzzer, ETHPLOIT, which exploits three types of complex and unsolvable constraints when using symbolic execution in smart contracts and blockchain attribute vulnerabilities related to timestamps and block numbers. Related technologies: static taint analysis, detected EVM environments, and dynamic seeding strategies. Seeds refer to the parameters and variables that are fed back when running a transaction. Updating the seeds in real time can provide precise parameters. The use of taint analysis and dynamic seeding strategies can improve the efficiency of generating valid inputs. Compared to other vulnerability detectors, ETHPLOIT can detect many types of security vulnerabilities.

Ashraf *et al.* [67] proposed a new technology, GasFuzzer. It consists of two policies. The gas-Greedy strategy is based on the insight that the Gas consumption executing the transaction provides lightweight information about the executed program code to process the blockchain state of the associated smart contract. GasFuzzer uses this information to eliminate trades that further generate mutant trades. Experiments show that GasFuzzer can detect more Exceptions Disorder security vulnerabilities than ContractFuzzer, the previous most advanced black-box technology, without compromising its ability to detect other types of security vulnerabilities. The novelty of the gas-leveling strategy is that it develops a new test data adequacy standard and uses it to guide the generation of variation trades with lower Gas quotas. Experiments show that this strategy is effective in detecting Exceptions Disorder security vulnerabilities that are omitted from the above experiments. Through this work, it can be concluded that by focusing on high-cost transactions and manipulation of Gas quotas, the fuzzy testing process for some of the most serious security vulnerabilities that can arise in smart contracts can be significantly improved.

Zhou *et al.* [68] proposed a SMARTGIFT method that learns from the transaction records of real-world smart contracts in order to generate practical test inputs for new smart contracts being tested. A prototype implementation of the approach utilizes BERT pre-trained model to embed function signatures, allowing efficient search of similar functions from large datasets of functions implemented in smart contracts deployed in the real world. With 66,528 transaction records collected, SMARTGIFT was able to generate relevant test inputs for about 77% of smart contract functions, superior to baseline fuzziness testing methods. Experiments further show that the input generated by SMARTGIFT is complementary to the input obtained through the fuzziness test. We further evaluated the potential of detecting smart contract vulnerabilities using the test input generated by SMARTGIFT. By providing test input generated by SMARTGIFT to ContractFuzzer, a state-of-the-art tool, 131 of the 154 smart contract vulnerabilities from the benchmark were correctly detected. These experimental results suggest that the actual input generated by SMARTGIFT is meaningful for testing smart contracts and demonstrating the potential to detect smart contract vulnerabilities.

Choi *et al.* [69] proposed SMARTIAN to enhance fuzzy testing of smart contracts through static and dynamic data flow analysis. First, the method statically analyzes the smart contract bytecode to predict which sequences of transactions will lead to valid tests and determine whether each transaction should meet certain constraints. This information is then passed to the fuzzy-testing phase and used to build the initial seed corpus. In fuzziness testing activities, lightweight dynamic data flow analysis is performed to collect data flow-based feedback to effectively guide fuzziness testing. You don't need source code to find errors in real-world smart contracts. The results showed that SMARTIAN was more effective than the most advanced tools available at finding known general security vulnerabilities in specific contracts. SMARTIAN also outperforms other tools in terms of code coverage.

Echidna and ContractFuzzer are representative works of the early use of fuzzing for smart contract vulnerability mining, but the fuzzing strategies adopted by Echidna and ContractFuzzer are relatively rudimentary, and it is difficult to achieve high code coverage. ETHPLOIT uses three related technologies: static taint analysis, a detected EVM environment, and a dynamic seed strategy, which can effectively find blockchain attribute vulnerabilities related to timestamps and block numbers. ILF uses deep learning methods to generate better test case sequences to improve coverage, however, its solutions are only oriented to specific contracts, and rely on the contract's source code or calling interface specifications, making it difficult to expand on a large scale. Fuzzing testing can effectively exploit the security vulnerabilities of smart contracts, but there are still problems such as the inability to automate the testing of contracts without source code or calling interface information, poor effectiveness of the calling sequence, and the low accuracy of vulnerability detection. The use of fuzz testing to conduct large-scale testing of source-free smart contracts and how to find loopholes in testing more accurately are still urgent issues to be solved in smart contract fuzz testing.

2) SYMBOLIC EXECUTION

Symbolic execution is a traditional automatic vulnerability mining technology that is widely used in smart contract vulnerability mining. The symbolic execution engine provides a symbolic virtual running environment for the target code, abstracts the external input required by the program into symbolic values with variable values, and explores program branches as much as possible by continuously solving path constraints. The main idea of symbolic execution is to convert uncertain inputs during program execution into symbolic values to promote program execution and analysis.

Symbolic execution enables a more precise and comprehensive analysis of programs than fuzzing does. Rich information collected during symbolic execution can be used for vulnerability identification. However, under the influence of structures, such as branches and loops, symbolic execution often faces problems, such as path explosion, which results

in a large time overhead when analyzing large programs. Compared to traditional applications, smart contracts have smaller code sizes, fewer paths, and shorter lengths, which are more suitable for analysis using symbolic execution methods. However, because the constraints of smart contracts are transferred between different paths, the symbolic execution idea for traditional programs can only collect constraints on a single contract path in smart contracts, and not global constraints, which is the symbolic execution of smart contracts. However, this approach presents challenges.

Luu *et al.* [70] proposed an automated contract vulnerability mining system, called *Oyente*. Oyente provides a streamlined contract symbol execution engine for developers to write contracts. Oyente includes detection solutions for four types of contract vulnerabilities, including conditional competition, timestamp dependency, unchecked return value, and reentrancy vulnerabilities. The detection of common vulnerabilities, such as integer overflow, integer underflow, and call stack overflow, has been added to the open source code of the Oyente system, and Oyente designed a validator to filter out false positives generated by the analyzer. Oyente is a contract vulnerability mining tool at the bytecode level. It dynamically explores the program control-flow graph in the process of symbol execution and detects contract vulnerabilities based on information such as path constraints and variable sources. Although some of Oyente's detection schemes are not perfect and the detected vulnerabilities are not sufficiently comprehensive, their pioneering work still provides good support for follow-up research.

In 2017, Consensys developed a symbolic execution engine called Mythril [71] that can analyze various EVM bytecode-based contracts. Mythril provides a wealth of extension interfaces, and developers can write custom vulnerability detection logic based on it. At present, Mythril officially provides 14 vulnerability detection functions, including integer overflow, arbitrary address writing, and timestamp dependence. Unlike Oyente, Mythril simulates a situation in which the contract is called multiple times in reality by means of multiple symbolic executions. This method avoids the error caused by initializing global variables to arbitrary symbolic values, and truly depicts the actual situation of contract execution. However, each round of symbolic execution must continue on all branch states generated by the previous round of symbolic execution, resulting in an exponential increase in the number of paths that need to be explored, resulting in a large time overhead.

The imitation-learning-based Fuzzer proposed in 2019 [72] (ILF) is a smart contract fuzzing solution based on symbolic execution and neural network, which is dedicated to generating better test cases and transaction sequences in smart contract fuzzing. Because smart contracts have global state, it is difficult to provide coverage guidance feedback for test cases with a single function to improve global coverage. Because the constraints in some functions are introduced by the global state variables, the values of these function variables may depend on the modification of other functions. Therefore, to improve global program coverage and achieve a better vulnerability mining effect, it is necessary to generate a reasonable calling sequence. In solving the problem of global constraints, symbolic execution can directly solve the solution space state of constraints, and the effect of symbolic execution is better than that of fuzzing. Therefore, the ILF's solution is to first generate a large number of excellent call sequences with the help of a symbolic execution engine, and then learn the characteristics of these excellent call sequences through neural networks to guide the fuzzing engine to generate excellent scheduling strategies.

Mossberg *et al.* [73] proposed Manticore, which is a dynamic symbolic execution engine for smart contracts. This improves the analysis efficiency and code coverage by specifying the variable values. Manticore abstracts the execution process of the contract into three states: ready, busy, and terminated, and transitions can be made between these three states. When a symbolic variable must be converted to one or more concrete values, a corresponding ready state is created and processed, terminating when the program exits or an exception occurs. Unlike other tools that analyze a single contract, Manticore can support the analysis of multiple contracts, which is equivalent to symbolizing the Ethereum world. Additionally, Manticore supports the symbolic execution of other binaries.

Wang *et al.* [74] proposed ContractWard to use machine learning technology to detect vulnerabilities in smart contracts. First, the binary characteristics are extracted from the simplified operation codes of smart contracts. Secondly, five machine learning algorithms and two sampling algorithms are used to construct the model. ContractWard was evaluated using 49,502 real-world smart contracts running on Ethereum. Experimental results demonstrate the effectiveness and efficiency of ContractWard.

Chen *et al.* [75] proposed DefectChecker, a symbol-based execution method and tool for detecting eight contract defects that could lead to bad behavior in smart contracts on the Ethereum blockchain platform. DefectChecker can detect contract defects from the bytecode of a smart contract. DefectChecker can detect contract defects from bytecode without the need for source code, and developers can use DefectChecker to check that the smart contracts they invoke are secure, even if the called contracts are not open source, which can make their contracts more secure.

Jin *et al.* [76] proposed a cross-platform open source framework called EXGEN for automatically generating exploits for smart contracts on EOS and Ethereum. EXGEN first converts ethereum or EOS contracts into intermediate representations (IR). EXGEN then generates symbolic attack contracts with partially sequential transactions, and then signs attack contracts with the target to find and resolve all constraints. Finally, EXGEN materializes all symbols to generate attack contracts that contain multiple transactions, and verifies the availability of the resulting contracts on the private chain by crawling values from the public chain. The evaluation showed that EXGEN successfully outperformed

teEther, the most advanced tool, in generating exploits for vulnerable Solidity contracts. The evaluation also showed that EXGEN was able to find 50 Zero-day vulnerabilities in 24 EOS smart contracts.

As one of the efficient and effective smart contract vulnerability detection methods, deep learning method has been studied for its fast detection speed and high detection accuracy. More recently, deep learning methods using convolutional neural networks (CNN) have been actively investigated to transform image classification from smart contracts to vulnerable or unassailable. However, the semantics and context of smart contracts can be ignored when they are simply converted into images and analyzed, leading to false detection alerts. To detect vulnerable smart contracts while preserving their semantics and context, Hwang *et al.* [77] proposed a new code-targeting CNN architecture called CodeNet. To improve the performance of CodeNet, a data preprocessor was designed to convert smart contracts into images while maintaining locality. According to the experimental results of various types of vulnerabilities, the proposed vulnerability detection method based on CodeNet has good detection performance and detection time compared with the most advanced vulnerability detection tools known to all.

In summary, the proposed schemes effectively improve the performance of symbolic execution mining vulnerabilities, but also significantly increase the computational time and resource overhead of analysis work, and cannot completely solve the problem of state-space explosion. Nevertheless, to maintain a balance between vulnerability mining efficiency and computational overhead, symbolic execution remains a relatively mainstream smart contract vulnerability mining solution.

3) FORMAL VERIFICATION

Formal verification is an important means of solving the security problem of smart contracts and an important research direction for smart contracts. The formal verification method is an effective means of deterministic verification of smart contracts. The concept, judgment and reasoning of smart contracts are transformed into smart contract models through formal language, which can eliminate the ambiguity and incompatibility of natural language. Formal verification tools are then used to model, analyze, and verify smart contracts, conduct semantic consistency testing, and automatically generate verified contract codes [78].

Hirai *et al.* [79] formally described Ethereum smart contract instructions. They used Isabelle/HOL to formally describe the instruction semantics in the Ethereum virtual machine, and manually proved the security of a program based on the description results. Because it supports only some of the instructions in Ethereum, it cannot describe the complete semantics of the contract.

Xiao *et al.* [80] proposed the smart contract specification language SPESC. The language defines some specifications of smart contracts that can support the collaborative design of contracts and provide technical support for multiple people to write smart contracts. The SPESC specification language is extremely simple to define smart contracts, and it helps users clarify the rights and obligations involved in the contract, just like writing real-world contracts in natural language. Park *et al.* [81] formally verified a smart contract with a deposit function in Ethereum, using the K framework. They first compile smart contracts to bytecode and then formalize and verify the bytecode. This method ensures that the function and security of smart contracts do not depend on the compiler's correctness.

Kalra *et al.* [82] designed ZEUS, which is an automatic formal verification tool that supports five security vulnerabilities in smart contracts. ZEUS converts the Solidity source code to the LLVM intermediate language, and uses XACML to write validation rules, and further uses the SeaHorn [83] validator for formal verification. Similar to previous solutions, the advantage of converting the Solidity code to the LLVM intermediate language is that it can use formal verification solutions for traditional programs. The disadvantage is that it may be difficult to express all the semantics of smart contract programs during the conversion process, particularly smart contracts. ZEUS has designed five security vulnerability detection rules that can determine the security of a target program in the process of formal verification. Antonino and Roscoe [84] abstracted the Solidity language based on the behavior of Ethereum smart contracts. Using this approach, Solidity's model checker *solidifier* was created. The process converts Solidity into Boogie, an intermediate verification language, which is then verified using the Corral boundary model checker [85] to verify the functionality of the smart contract.

Yang *et al.* [86] combined symbolic execution and a higher-order logic theorem to design a Formal Symbolic Process Virtual Machine of Ethereum (FSPVM-E), which is an extension of Curry Howard's isomorphic [87] application and can be used to verify the security and reliability of smart contracts at the source code level. The virtual machine FSPVM-E was developed based on the higher-order logic theorem proving in the formal proof assistant Coq. It uses the Coq theorem proving assistant to program the smart contract system, and uses the Hoare logic in Coq to verify the security attributes. The FSPVM-E can automatically execute Ethereum-based smart contracts, and scan smart contracts for possible security vulnerabilities.

Alqahtani *et al.* [88] proposed a formal method of model checking to verify the security of interactive smart contracts developed and controlled by different entities. The authors used the NuSMV model checker and behavioral interaction prioritization tool to model the behaviors and interactions of smart contracts to verify whether the smart contracts meet the requirements of system functionality. A case study of the American Petroleum Institute confirmed the applicability of this method and used it to verify the security of interactive smart contracts on the Fabric blockchain platform.

Nam and Kil [89] proposed a novel form verification technique by using ATL (Alternating-Time Temporal Logic)

model checks to analyze blockchain smart contracts. The given smart contract is first translated in the language of MCMAS, a state-of-the-art ATL model checker for multi-agent systems. If the attacker's contact information is given, it will also be simplified to MCMAS. Otherwise, the loosest attacker model is introduced and then encoded as MCMAS. In addition, the method adds an environment agent that manages their turn. Finally, verify the multi-agent system with the ATL properties that the developer wants to check. If MCMAS returns true, these properties are determined to be satisfied. Otherwise, MCMAS provides a counterexample that can go a long way toward helping developers to correct errors in contracts. In addition, three case studies are presented to show that this verification technique can be successfully applied to real-world smart contracts.

As blockchain technology evolves, the use of smart contracts is diversifying. Blockchain-based smart contracts are suitable for areas where integrity and transparency must be guaranteed with distributed ledger technology at its core. However, the system cannot be modified once deployed, so it is important to ensure that the system complies with the requirements and principles of the smart contract during the design phase. To solve this problem, Park *et al.* [90] used the UPPAAL model checking tool of time attribute to specify and verify the system to model and verify the Dutch auction trading system based on smart contract, and set time constraints for each template and state in the modeling process. Furthermore, by specifying the time constraint as a range, the price will fall at the correct time. Various scenarios at the end of the auction were supplemented by simulations and verified by using TCTL that all attributes derived from the functional requirements were met. Formal verification of UPPAAL proves that the system is accurate and error-free.

In summary, formal verification technology is widely used to detect whether a smart contract satisfies the security attribute rules defined by the contract designer, but it is not as widely used as fuzzing and symbolic execution in some common vulnerability detection. Compared to solutions based on other vulnerability mining technologies, most formal verification methods are currently not highly automated, and the detected vulnerabilities do not necessarily have an accessible program path. Therefore, the use of formal verification technology to mine general vulnerabilities in smart contracts still presents challenges.

4) OTHER TECHNOLOGIES

Zhou *et al.* [91] proposed a smart contract vulnerability mining method called SASC, based on static program analysis technology. Through the automatic analysis of the smart contract's source code, SASC determines the function call topology in the contract and the control flow characteristics inside the function, and detects the illegal use of variables such as blockchain timestamp and transaction source address (tx.origin), thereby flagging and reporting security vulnerabilities.

Tikhomirov *et al.* [92] designed a static analysis tool called SmartCheck for Ethereum smart contracts. SmartCheck works at the Solidity source code level of the smart contract, which further performs syntax and lexical analyses on the source code, and uses XML to describe the result of the abstract syntax tree after analysis. On this basis, Xpath [93] is used to check the security properties of smart contracts to detect loopholes in contracts, such as reentrancy, timestamp dependence, denial of service, and fund locking.

In 2020, Slither [94] launched a static analysis framework open-source project focused on smart contract program analysis. Based on the static analysis of the smart contract source code, the project designed two modules: printer and detector, which were used to analyze smart contracts. Its anomaly detection can detect as many as forty kinds of anomalies of smart contract, including a variety of security vulnerabilities. Slither created an intermediate language called SlitherIR, which implements all static program analysis work at the intermediate language level, helping extend the analysis framework to different high-level languages and types.

V. CHALLENGES AND DEVELOPMENT TRENDS OF BLOCKCHAIN SMART CONTRACTS

A. CHALLENGES OF BLOCKCHAIN SMART CONTRACT TECHNOLOGY

Smart contracts have several advantages. For example, an intermediate guarantor is no longer required for transactions, and both parties to the transaction no longer have to worry about trust issues, which will speed up contract verification and execution. A smart contract corresponds to a piece of program code, but once deployed in the blockchain, it cannot be changed. Because the blockchain can be regarded as a distributed database and smart contracts are deployed in the blockchain system, it also has the advantage of a distributed system, which can ensure the security and reliability of data. However, smart contract technology is still far from practical application and faces many challenges.

(1) The development language of contracts is still immature, and there is a lack of effective detection and processing methods for potential vulnerabilities. For example, the solidity development language of Ethereum lacks safe handling methods for problems such as function variables and operation symbols out of bounds, and most developers do not have enough semantic understanding of these development languages to use Turing machines flexibly, which can easily lead to security vulnerabilities in smart contracts. Since then, Ethereum smart contracts cannot be modified once deployed, and it is particularly difficult to solve the security problem in smart contracts. If the smart contract code design contains errors, there is no direct error-correction method after deployment into the chain. Therefore, it is necessary to design a method for modifying and ending the contract state. However,

this kind of method is contrary to the principle of ''code is law'' of Ethereum itself, or even if this method is designed, it is difficult to be recognized by the entire Ethereum node. If some nodes are not recognized, another blockchain is formed. Therefore, this type of security problem for smart contracts is one of the challenges it faces.

(2) The smart contract execution efficiency and data- processing ability of mainstream blockchain platforms are not high. Because mainstream blockchain platforms such as Ethereum and Hyperledger can be regarded as distributed databases, each node shares the data of the entire blockchain, resulting in the data-processing capability of the platform being no greater than that of a single node. Another problem is the serial execution of blockchain smart contract code leads to a low ability to process data. Several researchers have proposed sharding and multichannel solutions. These technical solutions are expected to be applicable to existing blockchain platforms. However, owing to the immaturity of existing technologies and their complex security situation, these technologies have not yet been applied to blockchain platforms. Therefore, applying these technologies to mainstream blockchain platforms to improve the execution efficiency and data processing capabilities of smart contracts is the second challenge.

(3) It is difficult for smart contracts to achieve storage expansion. The storage space of each block in the blockchain is fixed in size: accordingly, the size of the block is limited and cannot be expanded. The immutable nature of blockchain is that it can only add transaction data, and cannot modify or delete transaction data. Once the block data record is incorrect, it can only be solved by appending the correct record so that the complete modification process will be recorded. This mechanism is conducive to the decentralization of the system, but it severely limits the scalability of block storage. With the continuous increase in the number of transactions, the blockchain data will occupy the entire space, so it is urgent to realize scalable storage of smart contracts.

(4) The maintainability of a smart contract system is poor, and there are certain security risks. Because the smart contract cannot be modified once deployed, the maintainability of the code is poor, which causes serious hidden dangers to the system. If there is a security loophole in the smart contract deployed in the chain, there is no way to modify and supplement it, but to update the contract code and redeploy it, wasting storage space. In fact, smart contracts cannot be tampered with because the underlying characteristics of the blockchain are difficult to modify. Therefore, researchers can improve the maintainability of smart contracts by upgrading, iterating and deleting smart contracts.

B. DEVELOPMENT TREND OF BLOCKCHAIN SMART CONTRACT TECHNOLOGY

The operation of smart contracts based in the Ethereum is called the beginning of the blockchain 2.0 stage. Currently, an increasing number of blockchain platforms have begun to support the operation of smart contracts. Smart contracts are an important means of managing the digital assets in the blockchain. Smart contract technology has received increasing attention and has become a popular research topic. Therefore, we will make the following predictions regarding the development trends of smart contracts.

(1) Continuous improvement in smart-contract execution efficiency performance. At present, smart contracts are limited by the performance of blockchain systems and cannot handle data with complex logic and large throughput. Therefore, the performance of smart contracts must be improved urgently. Some researchers have proposed a second-layer expansion chain method, which creates an off-chain execution environment for smart contracts through trusted hardware, records transaction payments and contract execution results on the chain, separates the execution of smart contracts from the consensus process, and guarantees the efficiency and privacy of smart contracts. Therefore, continuous optimization and improvement based on this scheme is important ways to improve the efficiency of smart contract execution.

(2) Overall improvement in smart-contract security performance. The security performance of smart contracts is affected by multilevel factors, such as cryptography, consensus mechanisms, and smart contracts. Security problems cannot be well resolved from only a single aspect of smart contracts. Existing research has failed to fully and comprehensively solve the security problems faced by blockchain. Therefore, considering multi-level and multi-faceted influencing factors will be an important reference for designing a more secure and perfect blockchain security protection scheme from a global perspective.

(3) Scale of formal verification of smart contracts and unification of verification tools. At present, formal verification tools for smart contract source code and Ethereum bytecode are generally in the initial stage and lack large-scale applications. Many methods can only detect a single category or a few vulnerabilities, and require considerable manpower. The vulnerabilities that these vulnerability detection tools can detect are different and there is an overlap in the detectable vulnerabilities between some tools, which makes the detection of contract vulnerabilities particularly complex and extremely inconvenient. Therefore, the formal verification of smart contracts needs to move towards the direction of scale and the unification of vulnerability security detection and verification tools to improve detection efficiency and reduce detection costs.

(4) Cryptographic security technology is used to meet the challenges of quantum computing. The premise of the birth of blockchain technology is that cryptography technology is safe and reliable, but there are some loopholes, and there is also the possibility of cracking in the cryptography technology, which poses a huge challenge to the security of the blockchain system. The concept of quantum computing poses a challenge to the security of blockchain, especially the quantum computing Shor algorithm proposed in 1994, which is a great threat to encryption algorithms such as DES. Once the cryptography technology on which the blockchain relies is cracked, the security and privacy of the blockchain system will not be guaranteed and blockchain technology will lose its most important advantages. Therefore, designing a trustworthy cryptographic security technology under the challenges of quantum computing will become the focus of blockchain technology development.

(5) Coordination and integration of smart contracts and traditional laws. Currently, smart contracts are in the stage of complementing and cooperating with traditional contracts. In the future, on the one hand, it is necessary to strengthen smart contracts' understanding of the law, establish smart contract review standards, reduce errors, and make smart contracts within the legal review standards. On the other hand, it is also necessary to supplement existing laws to clarify the various situations of smart contracts and the exact meaning expressed by the parties to the transaction to avoid difficult accountability afterwards. Through these two aspects, the purpose of strengthening the coordination and integration of smart contracts and traditional laws is achieved.

VI. CONCLUSION

Smart contract technology is a new stage in the development of the blockchain technology. It is widely used, not only in the field of financial transactions, but also in many aspects of social life. Despite the advantages of smart contract technology, it is still in the early stages of development and there are still many issues to be resolved.

This paper introduces the concepts related to blockchain smart contracts, and then focuses on the current difficulties of smart contracts and the progress of solutions to these difficulties. First of all, it briefly introduces the development process of blockchain, and then focuses on the research progress of smart contracts in blockchain 2.0 stage. To accurately understand the concept of smart contracts, this paper provides a formal definition of smart contracts and elaborates on the working mechanism of smart contracts, including the main body of smart contracts, data loading methods, and contract execution environment. Secondly, it summarizes the current difficulties faced by smart contracts, such as low execution efficiency, difficult expansion of data storage, easy disclosure of privacy, vulnerability to potential attacks, etc. Several related studies have been conducted in response to these problems. For example, (1) using scalable capacity expansion solutions, such as increasing block capacity, directed acyclic graphs, and sharding, to solve the efficiency and storage performance problems of smart contracts; (2) the combination of zero knowledge proof technology and trusted execution environment to improve the privacy protection ability of smart contracts; and (3) using fuzzing, symbolic execution, formal verification and other technologies to mine the potential security vulnerabilities of smart contracts and reduce their security risks. Finally, the challenges faced by blockchain smart contract technology are analyzed, and future development trends are discussed.

REFERENCES

[1] R. Böhme, N. Christin, B. Edelman, and T. Moore, "Bitcoin: Economics, technology, and governance," *J. Econ. Perspect.*, vol. 29, no. 2, pp. 213–238, 2015.

[2] M. Nofer and M. Gomber, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, pp. 183–187, Mar. 2021.

[3] J. L. Zhao, S. Fan, and J. Yan, "Overview of business innovations and research opportunities in blockchain and introduction to the special issue," *Financial Innov.*, vol. 2, no. 1, pp. 1–7, Dec. 2016.

[4] G. Wood. (2014). *Ethereum: A Secure Decentralised Generalised Transaction Ledger, Ethereum Project Yellow Paper*. [Online]. Available: https://www.mendeley.com/catalogue/db13375c-94f2-394a-a770-a51001b0404d/

[5] D. Efanov and P. Roschin, "The all-pervasiveness of the blockchain technology," *Proc. Comput. Sci.*, vol. 123, pp. 116–121, Jan. 2018.

[6] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, no. 9, pp. 1–21, Sep. 1997.

[7] P. He, G. Yu, Y. F. Zhang, and Y. B. Bao, "Survey on blockchain technology and its application prospect," *Comput. Sci.*, vol. 44, no. 4, pp. 1–7, Apr. 2017.

[8] Y. Yuan and F. Y. Wang, "The state of the art and future trends," *Acta Automatica Sinica*, vol. 42, no. 4, pp. 481–494, 2016.

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

[10] J. Fan, L.-T. Yi, and J.-W. Shu, "Research on the technologies of byzantine system," *J. Softw.*, vol. 24, no. 6, pp. 1346–1360, Jan. 2014.

[11] B. Hou and F. Chen, "A study on nine years of bitcoin transactions: Understanding real-world behaviors of bitcoin miners and users," in *Proc. IEEE 40th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Nov. 2020, pp. 1031–1043.

[12] X. J. Cai, "The principle of blockchain and its core technology," *J. Comput.*, vol. 44, no. 1, pp. 84–131, Jan. 2021.

[13] V. Buterin, *A Next-Generation Smart Contract and Decentralized Application Platform*. Zug, Switzerland: Etherum, Jan. 2014, pp. 1–36.

[14] L. Savron, "How blockchain technology could change our lives," *Ursidae, Undergraduate Res. J. Univ. Northern Colorado*, vol. 8, no. 1, p. 10, Apr. 2019.

[15] L. Zhang, B. X. Liu, R. Y. Zhang, B. X. Jiang, and Y. J. Liu, "Overview of blockchain technology," *Comput. Eng.*, vol. 45, no. 5, pp. 1–12, May 2019.

[16] S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7, pp. 50759–50779, 2019.

[17] L. Ouyang, S. Wang, Y. Yuan, X. Ni, and F. Y. Wang, "Smart contracts: Architecture and research progresses," *Acta Automatica Sinica*, vol. 45, no. 3, pp. 445–457, 2019.

[18] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart contract templates: Foundations, design landscape and research directions," 2016, *arXiv:1608.00771*.

[19] F. Zhang and E. Cecchetti, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 270–282, doi: 10.1145/2976749.2978-326.

[20] X. L. Cao, J. H. Zhang, and B. Liu, "Review on security, privacy, and performance issues of blockchain," *Comput. Integr. Manuf. Syst.*, vol. 27, no. 7, pp. 2078–2094, Jul. 2021.

[21] S. R. M. Sekhar, G. M. Siddesh, S. Kalra, and S. Anand, "A study of use cases for smart contracts using blockchain technology," *Int. J. Inf. Syst. Social Change*, vol. 10, no. 2, pp. 15–34, Apr. 2019.

[22] L. V. Antonio, "Smart contracts: A review of security threats alongside an analysis of existing solutions," *Entropy*, vol. 22, no. 2, pp. 203–232, Feb. 2020.

[23] P. W. Shor, "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer," *SIAM J. Comput.*, vol. 26, no. 5, pp. 1484–1509, 1997.

[24] N. Atzei, "A survey of attacks on Ethereum smart contracts (SoK)," in *Proc. 6th Int. Conf. Princ. Secur. Trust*, vol. 10204, Apr. 2017, pp. 164–186, doi: 10.1007/978-3-662-54455-6_8.

[25] Y. D. Ni, C. Zhang, and T. T. Yin, "A survey of smart contract vulnerability research," *J. Cyber Secur.*, vol. 5, no. 3, pp. 78–99, May 2020.

[26] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," *Distrib. Comput.*, vol. 33, nos. 3–4, pp. 209–225, Jul. 2019.

[27] K. Wüst, S. Matetic, S. Egli, K. Kostiainen, and S. Capkun, "ACE: Asynchronous and concurrent execution of complex smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2020, pp. 587–600, doi: 10.1145/3372297.3417243.

[28] W. F. Silvano and R. Marcelino, "Iota tangle: A cryptocurrency to communicate internet-of-Things data," *Future Gener. Comput. Syst.*, vol. 112, pp. 307–319, Nov. 2020.

[29] S. Das, V. J. Ribeiro, and A. Anand, "YODA: Enabling computationally intensive contracts on blockchains with Byzantine and Selfish nodes," 2018, *arXiv:1811.03265*.

[30] E. Sariboz, K. Kolachala, G. Panwar, R. Vishwanathan, and S. Misra, "Off-chain execution and verification of computationally intensive smart contracts," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2021, pp. 1–3. [Online]. Available: https://ieeexplore.ieee.org/document/9461142

[31] *CITA Technical White Paper*, CRYPTAPE, Hangzhou, China, 2019. [Online]. Available: https://www.chainnode.com/doc/3507

[32] F. L. Zhang and P. Y. Hou, "Framework for architecting smart contracts using microservices," *J. Softw.*, vol. 32, no. 11, pp. 3423–3439, Aug. 2021.

[33] M. Fang, Z. Zhang, C. Jin, and A. Zhou, "High-performance smart contracts concurrent execution for permissioned blockchain using SGX," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, Apr. 2021, pp. 1907–1912, doi: 10.1109/ICDE51399.2021.00175.

[34] Y. Chen, X. Li, J. Zhang, and H. Bi, "Multi-party payment channel network based on smart contract," *IEEE Trans. Netw. Service Manage.*, early access, Mar. 22, 2022, doi: 10.1109/TNSM.2022.3162592.

[35] J. Liu, P. Li, R. Cheng, N. Asokan, and D. Song, "Parallel and asynchronous smart contract execution," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 5, pp. 1097–1108, May 2022, doi: 10.1109/TPDS.2021.3095234.

[36] L. Luu, "A secure sharding protocol for open blockchains," in *Proc. CCS ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 17–30, doi: 10.1.145/2976749.2978389.

[37] A. E. Gencer, R. V. Renesse, and E. G. Sirer, "Service-oriented sharding with Aspen," 2016, *arXiv:1611.06816*.

[38] Zilliqa. *Next-Gen High Throughput Blockchain Platform*. Accessed: Mar. 4, 2022. [Online]. Available: https: //www.zilliqa.com

[39] T. Duong, "Multi-mode cryptocurrency systems," in *Proc. BCC 2nd ACM Workshop Blockchains, Cryptocurrencies, Contracts*, May 2018, pp. 35–46, doi: 10.1145/320523-0.3205237.

[40] L. Zhu, H. Yu, S. X. Zhan, W. W. Qiu, and Q. L. Li, "Research on high-performance consortium blockchain technology," *J. Softw.*, vol. 30, no. 6, pp. 1577–1593, Jun. 2019.

[41] *Taxa Website*. Accessed: Apr. 7, 2022. [Online]. Available: https://taxa.network/

[42] J. Wang, H. Yu, H. Zhen, and L. Han, "Access control methods of data sharing in cloud storage based on smart contract," *Netinfo Secur.*, vol. 21, no. 11, pp. 40–47, Nov. 2021.

[43] Y. Wang, H. Liu, J. Wang, and S. Wang, "Efficient data interaction of blockchain smart contract with Oracle mechanism," in *Proc. IEEE 9th Joint Int. Inf. Technol. Artif. Intell. Conf. (ITAIC)*, Dec. 2020, pp. 1000–1003, doi: 10.1109/ITAIC49862.2020.9338784.

[44] T. Kim, S. Lee, Y. Kwon, J. Noh, S. Kim, and S. Cho, "SELCOM: Selective compression scheme for lightweight nodes in blockchain system," *IEEE Access*, vol. 8, pp. 225613–225626, 2020, doi: 10.1109/ACCESS.2020.3044991.

[45] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "Hawk: The blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 839–858. [Online]. Available: https://ieeexplore.ieee.org/document/7546538/citations#citations

[46] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 270–282, doi: 10.1145/2976749.2978326.

[47] Microsoft. *The Coco Framework: Technical Overview*. Accessed: Feb. 8, 2022. [Online]. Available: https://github.com/Azure/coco-framework

[48] T. Hunt and Z. Zhu, "Ryoan: A distributed sandbox for untrusted computation on secret data," *ACM Trans. Comput. Syst.*, vol. 35, no. 4, pp. 1–32, Nov. 2017, doi: 10.1.145/3231594.

[49] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song, "Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts," in *Proc. IEEE Eur. Symp. Secur. Privacy (EuroS&P)*, Jun. 2019, pp. 185–200. [Online]. Available: https://ieeexplore.ieee.org/documen-t/8806762

[50] H. Kalodner and S. Goldfeder, "Arbitrum: Scalable, private smart contracts," in *Proc. SEC 27th USENIX Conf. Secur. Symp.*, Aug. 2018, pp. 1353–1370, doi: 10.5555/3277-203.3277305.

[51] ORIGO. (2019). *Origo Privacy Preserving Smart Contract blockchain*. [Online]. Available: https://origo.net-work/whitepaper

[52] H. Kopp, D. Mödinger, F. J. Hauck, and F. Kargl, "Cryptographic design of PriCloud, a privacy-preserving decentralized storage with remuneration," *IEEE Trans. Depend. Secure Comput.*, vol. 18, no. 4, pp. 1908–1919, Aug. 2021, doi: 10.1109/TDSC.2019.2942300.

[53] N. Kapsoulis, A. Psychas, G. Palaiokrassas, A. Marinakis, A. Litke, and T. Varvarigou, "Know your customer (KYC) implementati-on with smart contracts on a privacy-oriented decentralized architecture," *Future Internet*, vol. 12, no. 2, pp. 1–41, Feb. 2020.

[54] J. Hao, C. Huang, W. Tang, Y. Zhang, and S. Yuan, "Smart contract-based access control through off-chain signature and on-chain evaluation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 69, no. 4, pp. 2221–2225, Apr. 2022, doi: 10.1109/TCSII.2021.3125500.

[55] Z. Wan, Y. Zhou, and K. Ren, "zk-AuthFeed: Protecting data feed to smart contracts with authenticated zero knowledge proof," *IEEE Trans. Depend. Secure Comput.*, early access, Feb. 23, 2022, doi: 10.1109/TDSC.2022.3153084.

[56] J. Li, B. Zhao, and C. Zhang, "Fuzzing: A survey," *Cybersecurity*, vol. 1, no. 1, pp. 45–57, Dec. 2018.

[57] R. K. Prakash, "Hardiness sensing for susceptibility using American fuzzy lop," in *Proc. ITM Web Conf.*, Mar. 2021, Art. no. 01003, doi: 10.1051/itmconf/202-1370100.3.

[58] C. Aschermann, S. Schumilo, and T. Blazytko Jan. 2019), *REDQUEEN: Fuzzing With Input-to-State Correspondence. NDSS*. [Online]. Available: https://www.researchgate.net/public-ation/348915535_REDQUEEN_Fuzzing_with_Input-to-State_Correspondence

[59] P. Chen and H. Chen, "Angora: Efficient fuzzing by principled search," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 711–725. [Online]. Available: https://ieeexplore.ieee.org/d-ocument/8418633

[60] S. Gan, C. Zhang, X. Qin, X. Tu, K. Li, Z. Pei, and Z. Chen, "CollAFL: Path sensitive fuzzing," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 679–696. [Online]. Available: https://ieeexplore.ieee.org/document/8418631

[61] S. Rawat, V. Jain, A. Kumar, L. Cojocar, C. Giuffrida, and H. Bos, "VUzzer: Application-aware evolutionary fuzzing," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2017, pp. 1–14. [Online]. Available: https://www.ndsssymposium.org/nds-s2017/ndss-2017-programme/vuzzer-application-aware-evolutionary-fuzzing

[62] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," in *Proc. 40th Int. Conf. Softw. Eng., Companion*, May 2018, pp. 65–68. [Online]. Available: https://ieeexplore.ieee.org/docume-nt/8449446

[63] J.-W. Liao, T.-T. Tsai, C.-K. He, and C.-W. Tien, "SoliAudit: Smart contract vulnerability assessment based on machine learning and fuzz testing," in *Proc. 6th Int. Conf. Internet Things: Syst., Manage. Secur. (IOTSMS)*, Oct. 2019, pp. 458–465. [Online]. Available: https://ieeexplore.ieee.org/document/8939256

[64] (Mar. 2018). *Echidna. A Smart Fuzzer for Ethereum. Trail Bits Blog*. [Online]. Available: https://blog.trai-lofbits.com/2018/03/0-9/echidna-a-smart-fuzzer-forethereum/

[65] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 259–269. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9000089

[66] Q. Zhang, Y. Wang, J. Li, and S. Ma, "EthPloit: From fuzzing to efficient exploit generation against smart contracts," in *Proc. IEEE 27th Int. Conf. Softw. Anal., Evol. Reeng. (SANER)*, Feb. 2020, pp. 116–126. [Online]. Available: https://ieeexplore.ieee.org/document/9054822

[67] I. Ashraf, X. Ma, B. Jiang, and W. K. Chan, "GasFuzzer: Fuzzing ethereum smart contract binaries to expose gas-oriented exception security vulnerabilities," *IEEE Access*, vol. 8, pp. 99552–99564, 2020, doi: 10.1109/ACCESS.2020.2995183.

[68] T. Zhou, K. Liu, L. Li, Z. Liu, J. Klein, and T. F. Bissyande, "SmartGift: Learning to generate practical inputs for testing smart contracts," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2021, pp. 23–34, doi: 10.1109/ICSME52107.2021.00009.

[69] J. Choi, D. Kim, S. Kim, G. Grieco, A. Groce, and S. K. Cha, "SMARTIAN: Enhancing smart contract fuzzing with static and dynamic dataflow analyses," in *Proc. 36th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2021, pp. 227–239, doi: 10.1109/ASE51524.2021.9678888.

[70] S. Badruddoja, R. Dantu, Y. He, K. Upadhayay, and M. Thompson, "Making smart contracts smarter," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, May 2021, pp. 254–269. [Online]. Available: https://ieeexplor e.ieee.org/document/9461148

[71] B. Mueller. *Mythril-Reversing and Bug Hunting Framework for the Ethereum Blockchain*. Accessed: Apr. 18, 2022. [Online]. Available: https://pypi.org/project/mythril

[72] J. He, M. Balunovim, N. Ambroladze, P. Tsankov, and M. Vechev, "Learning to fuzz from symbolic execution with application to smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 531–548, doi: 10.1145/3319535.3363230.

[73] M. Mossberg, F. Manzano, E. Hennenfent, A. Groce, G. Grieco, J. Feist, T. Brunson, and A. Dinaburg, "Manticore: A user-friendly symbolic execution framework for binaries and smart contracts," in *Proc. 34th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2019, pp. 1186–1189. [Online]. Available: https://ieeexplore.iee e.org/document/8952204

[74] W. Wang, J. Song, G. Xu, Y. Li, H. Wang, and C. Su, "ContractWard: Automated vulnerability detection models for Ethereum smart contracts," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1133–1144, Apr. 2021, doi: 10.1109/TNSE.2020.2968505.

[75] J. Chen, X. Xia, D. Lo, J. Grundy, X. Luo, and T. Chen, "DEFECTCHECKER: Automated smart contract defect detection by analyzing EVM bytecode," *IEEE Trans. Softw. Eng.*, early access, Jan. 27, 2021, doi: 10.1109/TSE.2021.3054928.

[76] L. Jin, Y. Cao, Y. Chen, D. Zhang, and S. Campanoni, "EXGEN: Cross-platform, automated exploit generation for smart contract vulnerabilities," *IEEE Trans. Depend. Secure Comput.*, early access, Jan. 7, 2022, doi: 10.1109/TDSC.2022.3141396.

[77] S.-J. Hwang, S.-H. Choi, J. Shin, and Y.-H. Choi, "CodeNet: Code-targeted convolutional neural network architecture for smart contract vulnerability detection," *IEEE Access*, vol. 10, pp. 32595–32607, 2022, doi: 10.1109/ACCESS.2022.3162065.

[78] J. Zhu, K. Hu, and B. J. Zhang, "Review on formal verification of smart contract," *Acta Electronica Sinica*, vol. 49, no. 4, pp. 792–804, Nov. 2021.

[79] Y. Hirai, "Defining the ethereum virtual machine for interactive theorem provers," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Nov. 2017, pp. 520–535, doi: 10.1007/978-3-319-70278-0_33.

[80] X. He, B. Qin, Y. Zhu, X. Chen, and Y. Liu, "SPESC: A specification language for smart contracts," in *Proc. IEEE 42nd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2018, pp. 132–137. [Online]. Available: https://ieeexplore.ieee.org/docume-nt/8377649

[81] D. Park, "End-to-end formal verification of Ethereum 2.0 deposit smart contract," in *Proc. Int. Conf. Comput. Aided Verification*, Jul. 2020, pp. 151–164, doi: 10.1007/978-3-030-53288-8_8.

[82] S. Kalra, "ZEUS: Analyzing safety of smart contracts," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Jan. 2018, pp. 1–12, doi: 10.1472-2/NDSS.2018.23092.

[83] SeaHorn. *A Verification Framework*. Accessed: Mar. 5, 2022. [Online]. Available: https://seahorn.github.io

[84] P. Antonino and A. E. Roscoe, "Form alising and verifying smart contracts with Solidifier: A bounded model checker for Solidity," 2020, *arXiv:2002.02710*.

[85] A. Lal, "Corral: A solver for reachability modulo theories," in *Proc. 24th Int. Conf. Comput. Aided Verification*, Jul. 2012, pp. 427–443, doi: 10.1007/978-36.42-31424-732.

[86] Z. Yang, H. Lei, and W. Qian, "A hybrid formal verification system in coq for ensuring the reliability and security of Ethereum-based service smart contracts," *IEEE Access*, vol. 8, pp. 21411–21436, 2020.

[87] P. Wadler, "Propositions as types," *Commun. ACM*, vol. 58, no. 12, pp. 75–84, Nov. 2015.

[88] S. Alqahtani, X. He, R. Gamble, and P. Mauricio, "Formal verification of functional requirements for smart contract compositions in supply chain management systems," in *Proc. Annu. Hawaii Int. Conf. Syst. Sci.*, 2020, pp. 1–10. [Online]. Available: http://hdl.handle.net/10125/64392

[89] W. Nam and H. Kil, "Formal verification of blockchain smart contracts via ATL model checking," *IEEE Access*, vol. 10, pp. 8151–8162, 2022, doi: 10.1109/ACCESS.2022.3143145.

[90] W. S. Park, H. Lee, and J.-Y. Choi, "Formal modeling of smart contract-based trading system," in *Proc. 23rd Int. Conf. Adv. Commun. Technol. (ICACT)*, Feb. 2021, pp. 48–52, doi: 10.23919/ICACT51234.2021.9370462.

[91] E. Zhou, S. Hua, B. Pi, J. Sun, Y. Nomura, K. Yamashita, and H. Kurihara, "Security assurance for smart contract," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5. [Online]. Available: https://ieeexplore.ieee.org/document/8328743

[92] S. Tikhomirov, "Smar tCheck: Static analysis of Ethereum smart contr acts," in *Proc. 1st Int. Workshop IEEE Comput. Soc.*, May 2018, pp. 9–16. [Online]. Available: http s://ieeexplore.ieee.org/document/8445052

[93] *Xpath Cover Page—W3C*. Accessed: Mar. 24, 2022. [Online]. Available: https://www.w3.org/TR/xpath/all/

[94] (2020). *Crytic/Slither*. [Online]. Available: https://github.com/crytic/slither

CANGHAI WU born in 1979. She received the master's degree from Jiangxi Normal University, in 2007. She is currently an Associate Professor with the Software College, Jiangxi Agricultural University. Her main research interests include blockchain, parallel distributed processing, and cloud computing.

JIE XIONG is currently pursuing the master's degree with Jiangxi Agricultural University, China. His current research interests include blockchain and agricultural informatization.

HUANLIANG XIONG was born in 1977. He received the Ph.D. degree in computer software and theory from the Department of Computer Science, Tongji University, in 2017. He is currently an Associate Professor with Jiangxi Agriculture University, where he is the Faculty of the Software College. His research interests include blockchain, parallel distributed processing, and cloud computing.

YINGDING ZHAO was born in 1965. He received the Ph.D. degree in electronic physics and devices from Tsinghua University, in 1992. He is currently a Professor and the Dean with the School of Software, Jiangxi Agricultural University, China. He has presided over or participated in more than ten scientific research projects, such as ''Network water army identification research in social networks'' and ''Design and implementation of navigation and positioning system based on Beidou satellite.'' He has published more than 20 academic papers. His research interests include agricultural information technology, networks, and information security.

WENLONG YI received the M.S. and Ph.D. degrees in computer science and engineering from Saint Petersburg Electrotechnical University ''LETI,'' Russia, in 2008 and 2018, respectively. He is currently an Associate Professor with the School of Software, Jiangxi Agricultural University, China. His research interests include distributed systems, the Internet of Things, and cryptography. He is a Senior Member of the China Computer Federation (CCF).