IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 20, 2019, accepted June 3, 2019, date of publication June 7, 2019, date of current version June 27, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2921624

A Survey on Security Verification of Blockchain Smart Contracts

JING LIU, (Member, IEEE), AND ZHENTIAN LIU
College of Computer Science, Inner Mongolia University, Hohhot 010021, China

Corresponding author: Jing Liu (liujing@imu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61662051.

ABSTRACT A smart contract is an agreement between two or more parties, which is executed by the computer code. The code does the execution without giving either party the ability to back out, so it ensures the trustless execution. The smart contract is one of the most important features in blockchain applications, which implements trusted transactions without third parties. However, with the rapid development, blockchain smart contracts have also exposed many security problems, and some attacks caused by contract vulnerabilities have led to terrible losses. In order to better deal with such dilemma, making a comprehensive survey about the security verification of blockchain smart contracts from major scientific databases is quite indispensable. Even though the significance of studying security verification of blockchain smart contracts is evident, it is really fresh yet. The major contributions of our survey work come from three aspects. First, after retrieving all-sided research studies, we select 53 most related papers to show the state-of-the art of this topic, where 20 papers focus on dealing with security assurance of blockchain smart contracts, and 33 papers focus on the correctness verification of blockchain smart contracts. Second, we propose a taxonomy toward the topic of security verification of blockchain smart contracts and discuss the pros and cons of each category of related studies. Third, through in-depth analysis of these studies, we come to know that the correctness verification of smart contracts based on the formal method has already become the more significant and more effective method to validate whether a smart contract is credible and accurate. So, we further present representative studies of formal verification of smart contracts in detail to demonstrate that using a formal method to validate blockchain smart contracts must have a promising and meritorious future.

INDEX TERMS Blockchain, formal method, security verification, smart contract, survey.

I. INTRODUCTION

As for the dilemma of trust crisis, transactions involving authentic third parties also have security risks or high costs. Recently, as a decentralized distributed ledger, the blockchain enables people to conduct secure and reliable transactions in an untrustworthy environment, which has been rapidly developed and widely deployed. Blockchain was born with the advent of Bitcoin, but now the blockchain is not limited to digital currency. It has significant achievements in economy, politics, industry, and society.

In the ecosystem of blockchain, the smart contract has become one of the most important features in practical applications. The smart contract was proposed by cryptographer Szabo [1] in 1997. It is defined as a digital agreement promised by contract participants. That is, it is a piece of code that can execute automatically on a computer. Due to the related technology limitation at that time, the development of smart contracts remained stagnant. But now the blockchain technology provides a natural execution environment for smart contracts which makes it popular again. Smart contracts have applied in many scenarios such as crowdfunding, voting, securities and medical research [2]. The most common deployment platform for smart contracts is Ethereum [3]. However, with the rapid development of smart contracts, the number of attacks is also growing. In June 2016, the DAO (the world's largest crowdfunding project deployed on the Ethereum) was attacked by hackers [4], causing more than 3 million ETH separated from the DAO resources pool. In September 2017, the security vulnerability appeared in the Ethereum multi-signature wallet Parity, which had resulted in more than 150,000 ETH (about $30 million) embezzlement.

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

And in April 2018, the BEC attack caused about $900 million stolen. Facing with such painful losses, how to ensure the security and reliability of smart contracts has attracted extensive attention.

The security and reliability of smart contracts include two dimensions. One is to regard smart contract as a static program that has not been put into use. The correctness of the program is a prerequisite for ensuring the security and reliability of the contract. The other one is the security issues that may arise during the execution of the contract. By considering these two dimensions in a comprehensive way, the security and reliability of smart contracts can be greatly improved. Therefore, we make an in-depth survey about the research results related to security verification of smart contracts in recent years, and analyze them from two aspects, the security assurance and the correctness verification. The major contributions of our survey work comes from three aspects. First, we select 53 most related papers, and make a synoptic overview. Second, we propose a detailed taxonomy towards these aspects and discuss the pros and cons of each category of related studies. Third, through in-depth analysis, we find the formal method has already become a more effective method to validate whether a smart contract is credible and accurate, which has a promising and meritorious future.

This paper is organized as follows. Section II presents the preliminary about blockchain technology and smart contracts. Section III gives a synoptic overview of the related papers. Section IV describes the security assurance of blockchain smart contracts. Section V discussed the correctness verification of blockchain smart contracts, where the formal verification method is illustrated in detail. Then, we conclude our work in the last section.

II. PRELIMINARIES

A. BLOCKCHAIN TECHNOLOGY

In 2008, Nakamoto [5] published a bitcoin white paper which marks the birth of the blockchain. Blockchain can be understood as a decentralized distributed ledger. It enables peer-to-peer transactions in a distributed environment that does not require mutual trust through hash, signature, consensus algorithms, time stamps, and incentive policies. Blockchain technology solves the problems of high cost, low efficiency, and insecure data in third parties. Moreover it is considered to be the fourth milestone in the history of human credit evolution [6].

In the blockchain, each distributed node uses the *Merkle Tree* structure to record the transactions that have occurred and encapsulates the transaction data into a block. These enormous transaction data constitutes the block body, while the other block information, such as the *Merkle Root Hash*, *Time*, and *Nonce* constitutes the block header. Blocks are connected in *Time* order by storing the hash value (calculated by SHA256) of the parent block shown in Figure 1. Based on such storage structure, if an attacker wants to tamper with a transaction in a block, all blocks after the block must be recalculated. So it is difficult for an attacker to achieve such a powerful computing power to tamper the information. By using the Proof-of-Work (PoW), Proof of Stake (PoS), or other consensus mechanisms, blockchain creates a trusted network, and all participants can collectively maintain block information without trusting a single node. Because of the decentralization, transparency and non-tampering integration, blockchain is no longer only used in Bitcoin. Blockchain technology proposes a solution to the double-spending problem and the Byzantine general problem. But, in theory, when a person reaches 51% of the computing power, he can still perform double spend attack, and as the miners' rewards decrease, the number of attacks continues to increase. Although the probability of successful attack is low, ETC, BCH and BTG have all suffered 51% of the power attack, causing huge losses. So the security of the blockchain requires more attention.

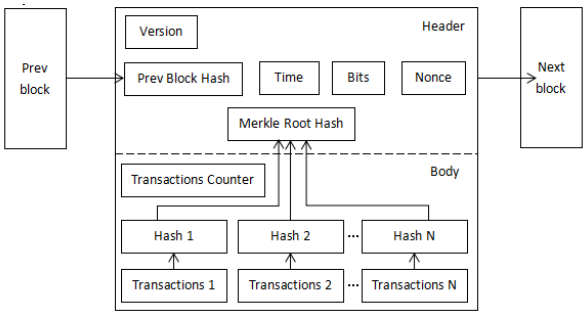

FIGURE 1. Block structure in blockchain system.

B. SMART CONTRACTS

Smart contract is a series of symbolic protocols that can be executed by contract participants. In 1997, Szabo [7] rewrote the smart contract publication as the *"formalization and security of public network relationships"*, which further improved the theoretical basis of smart contracts and the application of security protocols. However, smart contracts were not well developed due to the lack of relevant theoretical techniques and digital systems which support programmable smart contracts at that time. This is mainly reflected in three aspects. First, there are no digital assets that can be directly manipulated. Using an auto-executed contract to transfer the ''assets'' is the essence of a smart contract. But direct manipulation of real-world property, stocks, and other physical assets have become a major problem through computers. Second, the limitation of computational law [8] is a big problem. Computational law is an approach to automated legal reasoning focusing on semantically rich laws, regulations, contract terms, and business rules in the context of electronically-mediated actions. Traditional computational law focuses on analyzing and describing laws to help people understand and formulate real-life laws. However, the

emergence of smart contracts requires that laws be converted into executable code and achieving legal interaction between the machines. Third, lacking of a credible environment is the key point. For the auto-executed contract, the participants can only see the results after the execution, but do not know the intermediate process at all. However, there are several properties of the contract execution which need to be guaranteed, including contract execution correctness, non-tampering, and semantic equivalence of the contract with the participant intentions. Therefore, in such an environment where theory and technology are seriously scarce, the development of smart contracts has been stagnant.

In recent years, the decentralized blockchain technology has provided a trusted environment for smart contracts that does not require third-party intervention. Coupled with the rapid development of artificial intelligence, which has greatly promoted the progress and improvement of computational law, smart contracts have once again attracted people's attentions. Unfortunately, using vulnerabilities of smart contracts to attack contracts themselves has gradually increased. In 2018, Nikolic *et al.* [24] used the MAIAN analysis tool to perform a security analysis of nearly 1 million smart contracts, in 10 seconds per contract. Their analysis flags that 34,200 smart contracts are vulnerable. For further research, they randomly sampled 3,759 contracts in vulnerable contracts, and found that 3,686 smart contracts had a 89% probability of vulnerability. Therefore, the issue of security verification of smart contracts deserves imperative and comprehensive studies.

III. A SYNOPTIC OVERVIEW

To have a picture of the state-of-art of the security verification of blockchain smart contracts, we make an in-depth survey.

First, we make an effective survey towards the research about security verification of smart contracts. In our work, we adapt the *IEEE Xplore*, *ACM Digital Library*, *Elsevier ScienceDirect*, *Spinger*, *Web of Sciences*, *Engineering Village* as major scientific databases. We use *smart contract* together with *security* or *verification* as keywords. Until the end of Dec. 2018, 258 papers were appeared as the search results. According to the major contributions of these papers, a total of 53 related papers are selected as most related studies. Research papers that focused on the pure blockchain finance and blockchain law are not considered in our work. The amounts of related publications in recent years presented in Figure 2.

Second, we analyze all papers, and find two key points are shown obviously. On one hand, the topic of security verification of blockchain smart contracts is really fresh because the total number of papers is still small. While on the other hand, the research contributions of security verification of blockchain smart contracts have increased year by year, which means this research topic is imperative and promising. We propose a taxonomy towards the topic of security verification of blockchain smart contracts.

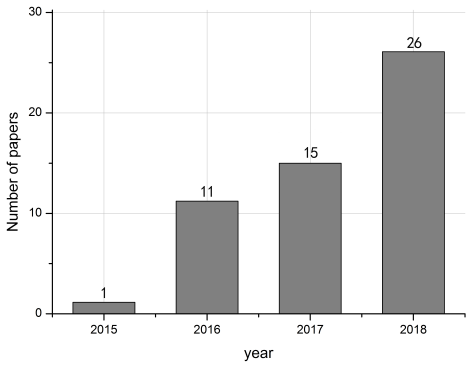

FIGURE 2. Amounts of related publications in recent years.

1) The aspect of security assurance of smart contracts in blockchain covered by 20 papers is classified into three categories, including environment security, vulnerability scanning, and performance impacts.
2) The aspect of the correctness verification of smart contracts covered by 33 papers is classified into two categories, including programming correctness and formal verification.

The classifications and major contributions of each paper is presented in Table 1.

IV. SECURITY ASSURANCE OF SMART CONTRACTS

With the rapid development of blockchain technology, smart contracts of great variety and diversity are flooding the market. How to maintain the safe and efficient execution of smart contracts has become a common pursuit. In this section, we discuss how to maintain the security of smart contracts from the aspects of the environment security, vulnerability scanning, and the performance impacts. Then, we discuss the pros and cons of each category of related studies.

A. ENVIRONMENT SECURITY

The correct execution of smart contracts requires a secure and reliable environment. In this section, we introduce the impact of environment security on smart contracts from two aspects: blockchain security and secure data source. The blockchain security provides a secure execution environment. The secure data source provides the credible data to ensure the secure execution. Below we discuss these two aspects in detail.

1) BLOCKCHAIN SECURITY

Several studies summarized common problems in blockchain security and depicted a macro blueprint for building a secure and reliable execution environment. Alharby and Moorsel [10] pointed out that the common risks in the blockchain including 51% of computing power, private key protection, criminal activities, double payment, and transaction information disclosure, etc. By analyzing the causes of the problems they provided some existing solutions. Li *et al.* [11] studied the execution process of smart contracts

TABLE 1. The classifications and major contributions of related papers.

Category	Focus	Ref. No.	Year	Major contribution
Environment Security	Blockchain security	[10]	2017	Identify the common risks in blockchains
		[11]	2017	Introduce the problems in smart contracts
	Secure data source	[12]	2016	Introduce Town Crier to ensure data security
Vulnerability Scanning	Comprehensive discussion	[13]	2016	Analyze the exist problems in smart contracts
		[9]	2017	Classify contract vulnerabilities
	The solution for a specific vulnerability	[14]	2017	Propose a mechanism for recycling stolen assets
		[15]	2018	Propose measures against eclipse-attack
		[16]	2018	Propose a tool to detect reentrant vulnerabilities
		[17]	2018	Judge termination to ensure the execution
		[18]	2018	Propose a tool called Osiris to detect integer bugs
		[19]	2018	Propose a method to deal with the callback bugs
	The solution for common vulnerabilities	[20]	2018	Propose a SmartInspect architecture to inspect the contracts
		[21]	2016	Design a symbolic execution tool - Oyente
		[22] [24]–[26]	2018	Design a verification tool (SolMet, MAIAN, ZEUS, Securify)
		[27]	2018	Propose six security design patterns
Performance Impacts	Execution method	[28], [29]	2017	Execute the independent smart contracts in parallel
Programming Correctness	Setting standards	[30]	2016	Write a lot of smart contracts to find errors
		[31]	2017	Give a programming model for smart contracts
		[32]	2016	Describe templates and agreements for contracts
		[33]	2016	Semi-automatic conversion of contracts
		[34]	2016	Propose the criteria for changing and revoking contracts
	Developing new contract language	[36]	2017	Improve upon existing crypto-currency languages
		[35], [39] [37], [38]	2016 2018	Idris, Flint, SmaCoNat, and Obsidian languages were designed to make the contract more secure
		[40]	2016	Propose a logical language to program contract
	Semantic analysis	[41] [42]	2017 2018	Present an executable formal specification of the KEVM
		[43]	2018	Design a SASC static analysis tool to analyse the semantic of the contract
		[44]	2017	Decompile and analyse the EVM code of the contract
		[45]	2018	Propose a novel semantic aware security auditing technique called S-gram for Ethereum
		[46]	2018	Introduce a framework called FSolidM which based on FSM
	Software engineering tools	[47]	2017	Devise specialized tools and techniques
		[48], [49]	2018	Call for the definition of a specific blockchain software engineering and design methods
Formal Verification	Comprehensive discussion	[53], [54]	2017	Propose that it is critical to introduce the formal method to the correct verification of smart contracts
	Program-based formal verification	[55] [59]	2016 2018	Use the F* proof assistant to verify the contracts
		[56] [57], [63]	2016 2018	Analyze the EVM Bytecode of the contracts statically
		[60], [61]	2018	Propose a novel formal symbolic process virtual machine (F-SPVM) to verify the properties of smart contracts
	Behavior-based formal verification	[64]	2018	Propose a runtime verification method
		[65]	2015	Use probabilistic formal models to verify contracts
		[66]	2018	Use the Promela language to verify smart contracts
		[67]	2018	Use the BIP framework to verify smart contracts

in the blockchain, and proposed two issues of transaction order dependency and timestamp dependency. The transaction was not executed in the correct order or the miner maliciously modified the block timestamp may affect the correctness of the smart contract. Simultaneously they suggested that using the OYENTE tool to detect smart contracts was a perfect solution.

2) SECURE DATA SOURCE

Smart contracts usually need to interact with external data sources, but they often fail to get external data over HTTPS. Therefore, Zhang *et al.* [12] introduced Town Crier (TC) to connect HTTPS data sources with smart contracts, which solved the problem of hindering the development of smart contracts due to the lack of trustworthy data feeds.

All the aspects above maintain a credible and secure environment for the execution of the smart contract, which reduce the possibility of successful external attacks.

B. VULNERABILITY SCANNING

Vulnerabilities caused by contract design defects can also cause huge losses. Vulnerability scanning is the study of vulnerabilities that have been discovered, which aimed at avoiding the same mistakes. It can help us discover potential vulnerabilities in the execution of contracts, which is important to improve the security and credibility of contracts.

To avoid the vulnerabilities, some studies systematically summarized the contract vulnerabilities caused by negligent design in recent years and analyzed the security risks in the contract [9], [13]. For example, Atzei *et al.* [9] summed up the contract vulnerabilities that have been discovered into 12 categories, explaining the reasons for the vulnerabilities. At present, reentrancy (DAO attack event), access control (Parity wallet stolen event), denial of services, bad randomness have caused huge losses in the discovered vulnerabilities, so it is important to avoid the recurrence of the vulnerability.

To solve the vulnerabilities, there are special solutions and common solutions that have been proposed. Below we discuss these two aspects.

1) THE SOLUTION FOR A SPECIFIC VULNERABILITY

Some studies have given different solutions to specific vulnerabilities. Bissias *et al.* [14] proposed a mechanism for recovering stolen assets for DAPP applications such as DAO, which greatly reduced the losses caused by attacks. Marcus *et al.* [15] proposed to eliminate some artifacts of Kademlia protocol, raising the threshold for attackers. This countermeasure has been added to geth1.8 (a standalone client of Go Ethereum) to against eclipse attacks. Liu *et al.* [16] proposed the ReGuard, a tool for detecting reentrant vulnerabilities, which dynamically identified reentrant vulnerabilities in contracts through fuzzing and automatically flaged reentrant errors generated. As the same, Shelly *et al.* [19] proposed a notion of Effectively Callback Free (ECF) objects, which dealt with the callback bugs in contracts, such as reentrancy. Le *et al.* [17] ensured the normal execution of the contract by calculating the execution model and statically proving the input conditions that guaranteed the termination or non-termination of the contract. Torres *et al.* [18] designed a framework to accurately find integer bugs in Ethereum smart contracts called Osiris. They evaluated the tool's performance on a large experimental dataset containing more than 1.2 million smart contracts, which found that 42,108 contracts contain integer bugs.

2) THE SOLUTION FOR COMMON VULNERABILITIES

In addition, some other work gave the solutions to common vulnerabilities. Bragagnolo *et al.* [20] proposed a SmartInspect architecture based on decompilation capabilities encapsulated in mirrors. It obtained unstructured information in the contract by decompiling the smart contract that had been deployed, so that they could introspect the current state of a smart contract instance. If any vulnerabilities were found, it can modify them without redepoly [23]. Wohrer and Zdun [27] proposed six security design patterns to reduce the possibility of the vulnerabilities. Besides, many tools had been designed to analyze the contracts, where the Oyente tool had achieved significant results. It extracted the control map from the EVM Bytecode of the contract and found potential vulnerabilities in the contract by executing the control map [21]. Similarly, there were also other verification tools, such as SolMet [22], MAIAN [24], ZEUS [25], Securify [26], Mythril [50], SmartDec [51], and Solgraph [52]. The first five tools analyzed the EVM Bytecode to check possible vulnerabilities during execution, and the last two tools analyzed the code of the original contract to find potential vulnerabilities.

Using vulnerability scanning method, we can more conveniently and comprehensively detect possible vulnerabilities in contracts when they are executing. So vulnerability scanning method is necessary for developing correct smart contracts.

C. PERFORMANCE IMPACTS

Performance can affect the security execution of smart contract. Due to the miner mechanism of blockchain and the poor Ethereum concurrency mechanism, the execution efficiency of smart contracts is very low. Only 10 to 20 transactions could be completed per second, leaving many complex financial transactions in the miners' pool for a long time. Through the analysis of Abdellatif and Brousmiche [67], we find that the longer the smart contract resides in the miners' pool, the more likely it is to be attacked successfully. Therefore, the performance impacts should be seriously considered.

Vukolic [28] argued that smart contracts which executed sequentially limited the attributes of the blockchain (such as confidentiality), so they proposed the idea of executing independent smart contracts in parallel. Then Dickerson *et al.* [29] proposed a new method of parallel execution of smart contracts, in which miners scheduled transactions and allowed non-conflicting contracts to execute in parallel. This method performed well on smart contract benchmarks, greatly speeding up contract execution efficiency.

D. DISCUSSION OF SECURITY ASSURANCE RELATED STUDIES

We summarize the achievements and weaknesses of above three aspects of related work as follows, which are used as guidance or suggestions for future studies.

Category: Environment Security

Achievement

- Common risks in blockchain have been discovered.
- Town Crier (TC) ensures smart contracts with secure data sources.

Weakness

- The solutions against risks in blockchain are still immature and lack of enough effectiveness.

Category: Vulnerability Scanning

Achievement

- We can learn lessons and avoid the same mistakes.
- A number of possible solutions to vulnerabilities are proposed, which makes the smart contracts more secure.

Weakness

- Only known vulnerabilities can be analyzed and unknown vulnerabilities cannot be discovered.
- The Vulnerability scanning is inefficient to analyze the huge smart contracts.

- Analysis is not comprehensive, it is easy to ignore some vulnerabilities.

Category: Performance Impacts

Achievement

- The idea of executing smart contracts in parallel is proposed together with some scheduling strategies for parallel execution.

Weakness

- The parallel execution method is still immature and the ethereum is still the most popular platform for smart contracts.

Through above analysis, we find that better and more meticulous solutions to conquer the security challenges of smart contracts are urgently needed. Currently, research on vulnerability scanning takes up the majority. Vulnerability scanning can effectively reduce potential vulnerabilities in contracts and ensure the secure execution of contracts. However, the vulnerability scanning tools are still immature, and unknown vulnerabilities cannot be found. So the vulnerability scanning technology needs further research.

V. CORRECTNESS VERIFICATION OF SMART CONTRACTS

In addition to security assurance, the correctness of the smart contracts deserves more attentions. Since there is no uniform programming specification for smart contracts and the limitations of the programming language, a variety of contract vulnerabilities have emerged, giving attackers opportunities to take advantage of them. So, in this section, we discuss how to write reliable smart contract from the following two aspects, which include programming correctness and formal verification.

A. PROGRAMMING CORRECTNESS

The essence of a smart contract is the computer code that can be executed automatically on the computer, so programming smart contract correctly is an important research direction. After summarizing the papers in recent years, we find that there are four ways of programming, including setting standards, developing new contract language, semantic analysis, and software engineering tools, which ensure the correctness of the contract.

1) SETTING STANDARDS

Setting standards is of great significance in regulating the programming of contracts. In order to find out the rules of programming, some studies focused on the large quantity of experiments. Delmolino *et al.* [30] used a large number of contracts to find common mistakes that were easy to occur during the writing process, and presented programming guidance to avoid these errors. Bartoletti and Pompianu [31] extensively analyzed the smart contracts related to Bitcoin and Ethereum applications, and provided a programming model for writing the correct contract. While, the other work started from the theory, they combined the knowledge of computational law to design the semantic framework of smart contracts [32], [33]. Through semantic transformation, the solidity contract was generated and finally tested. In addition, Marino and Juels [34] proposed a standard for changing and revoking contracts. Applying the framework to Ethereum can make it easier to modify or cancel the released smart contracts, which had great value in maintaining the correctness of contracts.

2) DEVELOPING NEW CONTRACT LANGUAGE

Developing new contract language is an effective way to write the correct smart contract. A variety of programming languages such as Idris [35], Simplicity [36], Obsidian [39], and Flint [37] have been proposed. These *Type-based* functional languages make the development process safer. Through the static analysis of smart contracts, we can more easily find vulnerabilities in the writing process which reduces the generation of contract errors and testing requirements. In addition to the programming languages mentioned above, Idelberger *et al.* [40] proposed a smart contract based on logic language, pointing out a new idea. The complement of logical language and programming language can make the design of smart contracts clearer. Regnath and Steinhorst [38] proposed a new programming language, SmaCoNat, which was a human-readable, security, and executable language. They converted programming language grammar into natural language sentences, such as directly giving names to variables instead of directly using memory addresses, which improved program readability. At the same time, to improve the security of the program, they reduced the possibility to repetitively alias logic and data structures by custom names.

3) SEMANTIC ANALYSIS

Semantic analysis is another important way to analyze and write a correct contract. Focused on modifying the EVM environment, Hildenbrandt *et al.* [41] presented an executable formal specification of the EVM's Bytecode stack-based language called KEVM. It provided a more security environment for the development of the smart contracts, and made the formal analysis of the smart contracts easier. They used the official Ethereum test suite to verify the correctness and performance of the KEVM, which all achieved better results. Daejun *et al.* [42] also proposed a formal verification tool for the EVM Bytecode, which adopted KEVM. Their verification tool had been used to verify various high-profile smart contracts including the ERC20, Ethereum Casper, and DappHub MakerDAO contracts. Focused on developing smart contracts, Zhou *et al.* [43] designed a SASC static analysis tool to generate a syntax topology map of the invocation relationships in smart contracts and mark the locations where risks and vulnerabilities may occur. Liu *et al.* [45] proposed a novel semantic aware security auditing technique called S-gram for Ethereum. They combined the N-gram language to model the static semantic labeling, and predicted potential vulnerabilities by identifying irregular token sequences. Mavridou and Laszka [46] introduced FSolidM, a framework rooted in rigorous semantics for designing contracts as FSM.

Also, they provided a tool to design the FSM model and verify the correctness of the contracts conveniently. Focused on the published smart contracts, Suiche [44] designed a decompiler to analyze the syntax and decompile the EVM binary code, so that the readable solidity code was generated. Through static and dynamic analysis of the decompiled code, vulnerabilities in contracts were discovered.

4) SOFTWARE ENGINEERING TOOLS

Software engineering tools provide well-accepted standards for the development of smart contracts, which formalized the process of writing the contract. Therefore, introducing the software engineering tools into the blockchain is conducive to the development of correct contracts. Porru *et al.* [47] proposed that designing specialized tools for blockchain-oriented software development and improving the testing activities would greatly facilitate the safe and reliable execution of smart contracts. Destefanis *et al.* [48] analyzed the Parity Wallet hack case, and found that the risks were mainly caused by a negligent programming activity. So they thought the blockchain oriented software engineering was necessary. Marchesi [49] listed several hacks successfully performed on blockchain, such as MtGox in 2014 (350 million US$), Bitfinex in 2016 (72 million US$), and Coincheck in 2017 (400 millionUS$). Through analysis, they proposed that the application of software engineering tools to blockchain software development might be crucial to the success of this new field. In short, software engineering tools can standardize the development process of smart contracts, thus developing smart contracts more efficient, and avoiding losses caused by negligence, such as batchoverflow.

All the methods mentioned above are useful for writing correct contracts. They regard a smart contract as a simple program and reduce the possible mistakes in the development process by standardizing and analyzing the contracts, which greatly improves the development efficiency of smart contracts.

B. FORMAL VERIFICATION

Formal methods provides a powerful technology for the correctness verification of smart contracts. At present, the use of formal methods to verify smart contracts has been widely recognized, and significant results have been achieved in practice. To solve the demands for high assurance contracts, the Ethereum community has turned to formal methods as well [70], [71].

Besides, using formal methods to validate smart contracts can provide a rigorous mathematical model for the verification of smart contracts. Through the analysis of the model, we can more easily discover the logic errors or other new vulnerabilities. The formal verification of smart contracts tends to be a great potential for development in the future [53], [54].

According to the research [53], we summarize the formal verification framework for smart contracts shown in FIGURE 3. The text agreement of a smart contract made by contract participants needs to fully express the intention

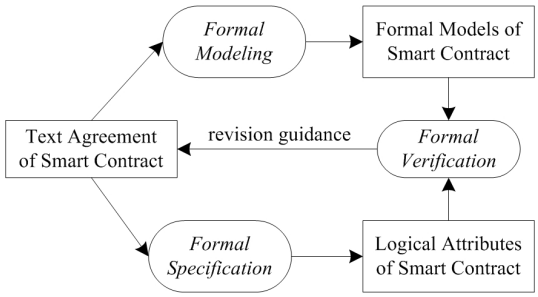

FIGURE 3. Formal verification framework for blockchain smart contracts.

of the makers and comply with the law. Then, Implementing formal specification and formal verification on above established contract, which is an iterative process. Through multiple behavioral modeling and attribute modeling of smart contracts, we can validate whether the attributes meet the contract requirement or not.

In our survey, we divide the formal verification method of smart contracts into two categories. One is program-based verification and the other one is behavior-based verification. Program-based verification treats smart contracts as codes, it translates the contract codes into formal languages and then identifies vulnerabilities in contracts. This is a verification method for static analysis programs. Behavior-based verification takes into account issues that may be encountered during the execution of smart contracts, such as improper operations and maliciously attacks. By constructing a formal model, it finds out the weak position of the contract, which is a way to dynamically analyze smart contracts. Below we discuss these two aspects in detail.

1) PROGRAM-BASED FORMAL VERIFICATION

The essence of a smart contract is the program that is executed on the blockchain. Verifying the correctness of the program is a key part of ensuring the safety and reliability of smart contracts.

There are both successful practical cases and theoretical studies on program-based verification of smart contracts. At the 3rd Global Blockchain Summit in 2017, a blockchain formal verification team of the University of Electronic Science and Technology presented the VaaS (Verification as a service) as a EOS formal verification platform. Except for the EOS blockchain platform, VaaS also supports other common platforms such as Ethereum and Fabric. The principle of VaaS platform is to translate programs written in the Solidity scripting language into the Coq code, thus establishing a standard formal model for smart contracts, and then verify the correctness of the smart contract by verifying the correctness of the Coq language. Similarly, Bhargavan *et al.* [55] proposed a verification method based on programming language. They translated the Solidity language into an F* language to check if the contract was correct. In most cases we can only get the binary code running on the Ethereum, and we cannot get the source code of the smart contract. So in the absence

of source code, they decompiled the binary files on Ethereum into F* language [58], and analyzed whether at least some of the attributes to be reached by the contract were satisfied. The complete F* language for any smart contract had not yet been implemented because of huge work. Moreover, for the same contract, it was so hard to verify whether the F* language translated by Solidity language consistented with the decompiled binary code. Furthermore, Grishchenko *et al.* [59] also used the F* language proof assistant to successfully validate the official Ethereum test suite.

Yang and Lei [60], [61] proposed a novel formal symbolic process virtual machine (FSPVM) to verify the reliability and security properties of smart contracts which based on Hoare logic in Coq. It used an extensible formal intermediate programming language Lolisa and the corresponding formally verified interpreter of Lolisa. After verification, they found that it supported the ERC20 standard and could solve the problems in higher-order logic theorem proving.

Hirai [56] proposed a formal verification method based on Ethereum Bytecode. In the process of translating one language into another, there may be cases where the meaning expressed in the translated language not be consistent with that before translation. So he proposed using the Isabelle/HOL to verify the binary Ethereum code. To illustrated his method, first, he obtained the binary instructions of a small contract called ''deed''. Then he translated the list of instructions into an AVL tree. Last, he analyzed the AVL tree in a theorem proving environment with the Isabelle proof assistant. This verification method is generic to all Ethereum smart contracts. Amani *et al.* [57] also used the Isabelle proof assistant to verify the binary Ethereum code. The principle of the method is to organize the bytecode sequences into linear code blocks and create a logic program, where each block is processed as a set of instructions. Each part of the verification is validated in a single trusted logical framework from the perspective of bytecode. Grishchenko *et al.* [63] focused on the EtherTrust [62], a framework for the static analysis of Ethereum smart contracts. It also analyzed the EVM Bytecode of the contracts statically together with a proof of soundness.

2) BEHAVIOR-BASED FORMAL VERIFICATION

Model checking is well adopt in behavior-based verification. It can conveniently model the interaction between the user and the program to verify whether the smart contract can interact with the user in a reliable and secure way. Some good examples of behavior-based formal validation are demonstrated as follows.

Ellul and Pace [64] proposed a runtime verification method. It was a novel state-based technique which ensured that the violating party provided insurance for correct behavior. They used the finite state machine to model the contracts, and this method had been partially implemented in a proof-of-concept tool ContractLarva. Their method refered to the methods proposed by Fenech *et al.* [68] and Gorin *et al.* [69], which validated the properties of the contract.

Then, Bigi *et al.* [65] combined game theory with formal methods and proposed a probabilistic formal model to verify smart contracts. They first analyzed the logic of the smart contract through game theory, then constructed a probabilistic formal model for the contract, and finally used the PRISM tool to verify the model. It was published in the early research stage of blockchain smart contracts, ensuring the safety and reliability of smart contracts through rigorous mathematical proofs. Similarly, Bai *et al.* [66] also proposed a model checking method. They used the Promela language to model a shopping contract and used the SPIN tool to verify whether the logic of the shopping contract was correct. These two papers validates from the perspective of user-program interaction behavior to determine whether the attributes are satisfied or not.

In addition to the above method, Abdellatif and Brousmiche [67] proposed a new verification method in 2018, which not only considered the interaction between users and programs, but also considered the interaction between programs and the environment, further advancing the development of using formal methods to verify smart contracts. They used the BIP (Behavior Interaction Priorities) framework to model components for smart contracts, blockchain, users, miners, trading pools, and attackers. Each component in the model was similar to an automatic state machine, which interacted through ports to enable dynamic interaction. Then they used the Statistical Model Checking (SMC) tool to verify whether the model satisfies a certain attribute. Specifically, the expected properties of the smart contract were described using the PB-LTL (Probabilistic Bounded Linear Time Logic) formula, which utilized probability to indicate the degree to which the model satisfied an attribute. This method simulated the whole process of smart contract execution (from being mined by miners to completing contracts) and modeled the attack's behavior. They found that attackers were the most likely to succeed in the process of interacting with the program, and that the likelihood of a successful attack would be significantly increased when the transaction was too long in the pending transaction pool.

C. DISCUSSION OF CORRECTNESS VERIFICATION RELATED STUDIES

We summarize the achievements and weaknesses of above three aspects of related work as follows, which are used as guidance or suggestions for future studies.

Category: Programming Correctness

Achievement

- Improving the accuracy of smart contract programs.
- Certain analytical tools improve the development efficiency of smart contracts, such as SASC.
- Traditional Software engineering tools are introduced which take instructive effects.

Weakness

- We do not have unified and authoritative programming framework for smart contracts.

- Newly developed languages have vulnerabilities and have not been put into practice.
- The converted smart contracts may differ from original smart contracts.
- Designing and implementing systematic smart contracts still require adaptive software engineering technologies.

Category: Formal Verification

Achievement

- We can use mathematical methods to model and rigorously verify the correctness of smart contracts.
- It can both statically analyze the logical structure of the contract and dynamically verify whether the contract is executed correctly.
- We could discover unknown vulnerabilities.

Weakness

- It is difficult to accurately model the smart contracts to be verified.
- Model checking or theorem proving technologies that consider user's behaviors are still in the infancy stage.

From the above analysis, we find that there are a lot of researches related to the correctness of smart contracts, which have achieved pretty good results. Furthermore, we come to know that the correctness verification of smart contracts based on formal method has already become the indispensable method to validate whether a smart contract is credible and accurate.

VI. CONCLUSION

As one of the most important features in blockchain systems, smart contracts have attracted much attentions but also have exposed many problems. The major contributions of our survey include three aspects.

First, we make a in-depth survey about the security verification of blockchain smart contracts and selected 53 most related papers. To show the state-of-art of this topic, we focused on two aspects, where 20 papers related to the security assurance and 33 papers related to the correctness verification.

Second, we propose a taxonomy towards such topic. More specifically, the security assurance aspects is classified into three categories, including environment security, vulnerability scanning, and performance impacts. While the correctness verification aspect is classified into two categories, including programming correctness and formal verification. We discuss the pros and cons of each category.

Third, through in-depth analysis of the related papers, we summarize the research status and point out the further research directions in smart contract security and correctness. The main points are as follows:

1) According to the growth trend of related papers, we can see that the security and correctness of smart contracts are getting more and more attention. We urgently need the more comprehensive methods to ensure the security and correctness of smart contracts, thereby reducing losses.
2) In the security of smart contracts, vulnerability scanning method is currently widely used and have achieved significant results. In future, we can continue to expand research in this direction. For example: discovering unknown vulnerabilities (try to infer whether there are vulnerabilities that have identical or similar principles to the known vulnerabilities) and optimizing vulnerability detection method, which avoids repeating a lot of vulnerability detection work and reduces errors or omissions.
3) In the correctness of smart contracts, more work is currently focused on programming correctness. Formulating the programming standards for smart contracts, designing a set of smart contract development processes, and improving the security awareness of programmers are all the future research directions.
4) Although the amounts of papers related to programming correctness is more, the growth trend of formal verification method is more obvious. Formal verification method is based on mathematical model and is more rigorous and reliable. Therefore, using formal verification method to verify smart contracts will be the trend of future research. In future, we can consider the following research directions: designing the more complete formal verification tools, combining formal verification methods with vulnerability analysis methods to complement each other, and visualizing the contract execution process using other formal modeling tools such as CPN (colored Petri Nets).

REFERENCES

[1] N. Szabo. *The Idea of Smart Contracts*. Accessed: May. 18, 2019. [Online]. Available: http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/idea.html
[2] D. Vujičić, D. Jagodić, and S. Randić, "Blockchain technology, bitcoin, and Ethereum: A brief overview," in *Proc. 17th Int. Symp. INFOTEH-JAHORINA (INFOTEH)*, Mar. 2018, pp. 21–23.
[3] V. Buterin. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. Accessed: May. 18, 2019. [Online]. Available: https://github.com/ethereum/wiki/wiki/White-Paper
[4] K. Finley. *Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform*. Accessed: May. 18, 2019. [Online]. Available: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/
[5] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: May. 18, 2019. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[6] Z. Zheng, S. Xie, H. Dai, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data, Big Data Congr.*, Honolulu, HI, USA, Jun. 2017, pp. 557–564.
[7] N. Szabo, "Formalizing and securing relationships on public networks," *First Monday*, vol. 2, pp. 1–9, Jun. 1997. doi: 10.5210/fm.v2i9.548.
[8] N. Love and M. Genesereth, "Computational Law," in *Proc. 10th Int. Conf. Artif. Intell. Law*, Jun. 2005, pp. 205–209. doi: 10.1145/1165485.1165517.
[9] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (SoK)," in *Proc. Int. Conf. Princ. Secur. Trust*, Apr. 2017, pp. 164–186. doi: 10.1007/978-3-662-54455-6_8.
[10] M. Alharby and A. V. Moorsel, "Blockchain-based smart contracts: A systematic mapping study," in *Proc. Int. Conf. Artif. Intell. Soft Comput.*, Aug. 2017, pp. 125–140. doi: 10.5121/csit.2017.71011.
[11] X. Li, P. Jiang, T. Chen, X. Luo, and Q. Wen, "A survey on the security of blockchain systems," *Future Gener. Comput. Syst.*, pp. 1–25, Aug. 2017. doi: 10.1016/j.future.2017.08.020.

[12] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi, "Town crier: An authenticated data feed for smart contracts," in *Proc. ACM Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1–20. doi: 10.1145/2976749.2978326.

[13] L. Anderson, R. Holz, A. Ponomarev, P. Rimba, and I. Weber, "New kids on the block: An analysis of modern blockchains," Jun. 2016, *arXiv:1606.06530*. [Online]. Available: https://arxiv.org/abs/1606.06530

[14] G. Bissias, B. N. Levine, and N. Kapadia, "Market-based security for distributed applications," in *Proc. New Secur. Paradigms Workshop*, Oct. 2017, pp. 19–34. doi: 10.1145/3171533.3171541.

[15] Y. Marcus, E. Heilman, and S. Goldberg. *Low-Resource Eclipse Attacks on Ethereum's Peer-to-Peer Network*. Accessed: May. 18, 2019. [Online]. Available: https://eprint.iacr.org/2018/236.pdf

[16] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, and B. Roscoe, "ReGuard: Finding reentrancy bugs in smart contracts," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng. Companion*, May 2018, pp. 65–68.

[17] T. C. Le, L. Xu, L. Chen, and W. Shi, "Proving Conditional Termination for Smart Contracts." in *Proc. 2nd ACM Workshop Blockchains, Cryptocurrencies, Contracts*, Jun. 2018, pp. 57–59.

[18] C. F. Torres, J. Schütte, and R. State, "Osiris: Hunting for integer bugs in ethereum smart contracts," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, Dec. 2018, pp. 19–34. doi: 10.1145/3274694.3274737.

[19] G. Shelly, A. Ittai, G.-G. Guy, M. Yan, R. Noam, S. Mooly, and Z. Yoni, "Online detection of effectively callback free objects with applications to smart contracts," in *Proc. ACM Program. Lang.*, Aug. 2018, pp. 1–28.

[20] S. Bragagnolo, H. Rocha, M. Denker, and S. Ducasse, "SmartInspect: Solidity smart contract inspector," in *Proc. Int. Workshop Blockchain Oriented Softw. Eng.*, Mar. 2018, pp. 9–18. doi: 10.1109/IWBOSE.2018.8327566.

[21] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Comput. Commun. Securit*, New York, NY, USA, Oct. 2016, pp. 254–269.

[22] P. Hegedus, "Towards analyzing the complexity landscape of solidity based ethereum smart contracts," in *Proc. IEEE/ACM 1st Int. Workshop Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, May 2018, pp. 35–39. doi: 10.1145/3194113.3194119.

[23] G. Bracha and D. Ungar, "Mirrors: Design principles for meta-level facilities of object-oriented programming languages," in *Proc. Int. Conf. Object-Oriented Program. Syst. Lang. Appl. ACM SIGPLAN Notices*, vol. 39, no. 10, Oct. 2004, pp. 331–344. doi: 10.1145/2854695.2854699.

[24] I. Nikolic, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proc. 34th Annu. Comput. Secur. Appl. Conf.*, Aug. 2018, pp. 1–15.

[25] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "ZEUS: Analyzing safety of smart contracts," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, Mar. 2018, pp. 1–15.

[26] P. Tsankov, A. Dan, D. Cohen, A. Gervais, F. Buenzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 67–82.

[27] M. Wohrer and U. Zdun, "Smart contracts: Security patterns in the Ethereum ecosystem and solidity," in *Proc. Int. Workshop Blockchain Oriented Softw. Eng. (IWBOSE)*, Mar. 2018, pp. 2–8.

[28] M. Vukolić, "Rethinking permissioned blockchains," in *Proc. ACM Workshop Blockchain Cryptocurrencies Contracts*, Apr. 2017, pp. 3–7.

[29] T. Dickerson, P. Gazzillo, M. Herlihy, and E. Koskinen, "Adding concurrency to smart contracts," in *Proc. ACM Symp. Principles Distrib. Comput.*, Jul. 2017, pp. 303–312.

[30] K. Delmolino, M. Arnett, A. Kosba, A. Miller, and E. Shi, "Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab," in *Financial Cryptography and Data Security*. Berlin, Germany: Springer, Feb. 2016, pp. 79–94. doi: 10.1007/978-3-662-53357-4_6.

[31] M. Bartoletti and L. Pompianu, "An empirical analysis of smart contracts: Platforms, applications, and design patterns," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.*, Mar. 2017, pp. 494–509. doi: 10.1007/978-3-319-70278-0_31.

[32] C. D. Clack, V. A. Bakshi, and L. Braine, "Smart contract templates: Foundations, design landscape and research directions," Aug. 2016, *arXiv:1608.00771*. [Online]. Available: https://arxiv.org/abs/1608.00771

[33] C. K. Frantz and M. Nowostawski, "From institutions to code: Towards automated generation of smart contracts," in *Proc. 1st Int. Workshops Found. Appl.*, Sep. 2016, pp. 210–215. doi: 10.1109/FAS-W.2016.53.

[34] B. Marino and A. Juels, "Setting standards for altering and undoing smart contracts," in *Proc. Int. Symp. Rules Rule Markup Lang. Semantic Web*, Jun. 2016, pp. 151–166. doi: 10.1007/978-3-319-42019-6_10.

[35] J. Pettersson and R. Edström. *Safer Smart Contracts Through Type-Driven Development*. Accessed: May. 18, 2019. [Online]. Available: http://publications.lib.chalmers.se/records/fulltext/234939/234939.pdf

[36] R. O'Connor, "Simplicity: A new language for blockchains," in *Proc. Workshop Program. Lang. Anal. Secur.*, New York, NY, USA, Oct. 2017, pp. 107–120.

[37] F. Schrans, S. Eisenbach, and S. Drossopoulou, "Writing safe smart contracts in flint," in *Proc. Conf. Companion 2nd Int. Conf. Art, Sci., Eng. Program.*, Apr. 2018, pp. 218–219.

[38] E. Regnath and S. Steinhorst, "SmaCoNat: Smart contracts in natural language," in *Proc. Forum Specification Design Lang.*, Sep. 2018, pp. 5–16. doi: 10.1109/FDL.2018.8524068.

[39] M. Coblenz, "Obsidian: A safer blockchain programming language," in *Proc. IEEE/ACM 39th Int. Conf. Softw. Eng. Companion*, May 2017, pp. 1–11.

[40] F. Idelberger, G. Governatori, R. Riveret, and G. Sartor, "Evaluation of logic-based smart contracts for blockchain systems," in *Proc. Int. Symp. Rules Rule Markup Lang. Semantic Web*, Jul. 2018, pp. 167–183. doi: 10.1007/978-3-319-42019-6_11.

[41] E. Hildenbrandt, M. Saxena, X. Zhu, N. Rodrigues, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu, and G. Rosu, "KEVM: A complete formal semantics of the ethereum virtual machine," in *Proc. IEEE 31st Comput. Secur. Found. Symp.*, Aug. 2017, pp. 204–217.

[42] P. Daejun, Z. Yi, S. Manasvi, D. Philip, and R. Grigore, "A formal verification tool for Ethereum VM Bytecode," in *Proc. 26th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, Nov. 2018, pp. 912–915. doi: 10.1145/3236024.3264591.

[43] 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.*, Feb. 2018, pp. 1–5. doi: 10.1109/NTMS.2018.8328743.

[44] M. Suiche, "Porosity: A decompiler for blockchain-based smart contracts bytecode," in *Proc. DEF*, Jul. 2017, pp. 1–30.

[45] H. Liu, C. Liu, W. Zhao, Y. Jiang, and J. Sun, "S-gram: Towards semantic-aware security auditing for Ethereum smart contracts," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Softw. Eng.*, Sep. 2018, pp. 814–819. doi: 10.1145/3238147.3240728.

[46] A. Mavridou and A. Laszka, "Designing secure Ethereum smart contracts: A finite state machine based approach," in *Proc. 22nd Int. Conf. Financial Cryptogr. Data Secur.*, Feb. 2018, PP. 1–15.

[47] S. Porru, A. Pinna, M. Marchesi, and R. Tonelli, "Blockchain-oriented software engineering: Challenges and new directions," in *Proc. IEEE/ACM Int. Conf. Softw. Eng. Companion*, Feb. 2017, pp. 169–171.

[48] G. Destefanis, M. Marchesi, M. Ortu, R. Tonelli, A. Bracciali, and R. Hierons, "Smart contracts vulnerabilities: A call for blockchain software engineering?" in *Proc. Int. Workshop Blockchain Oriented Softw. Eng.*, Mar. 2018, pp. 19–25. doi: 10.1109/IWBOSE.2018.8327567.

[49] M. Marchesi, "Why blockchain is important for software developers, and why software engineering is important for blockchain software (Keynote)," in *Proc. Int. Workshop Blockchain Oriented Softw. Eng.*, Mar. 2018, p. 1. doi: 10.1109/IWBOSE.2018.8327564.

[50] J. Honig, J. Asplund, and B. Mueller. *Mythril*. Accessed: May. 18, 2019. [Online]. Available: https://github.com/ConsenSys/mythril

[51] I. Sobolev, D. Alexander, and P. Yushchenko. *Smartcheck*. Accessed: May. 18, 2019. [Online]. Available: https://github.com/smartdec/smartcheck

[52] R. Revere and J. P. Antunes. *Solgraph*. Accessed: May. 18, 2019. [Online]. Available: https://github.com/raineorshine/solgraph

[53] D. Magazzeni, P. McBurney, and W. Nash, "Validation and verification of smart contracts: A research agenda," *Computer*, vol. 50, no. 9, pp. 50–57, 2017.

[54] S. Matsuo, "How formal analysis and verification add security to blockchainbased systems," in *Proc. Formal Methods Comput. Aided Design*, Oct. 2017, pp. 1–4. doi: 10.23919/FMCAD.2017.8102228.

[55] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, and S. Zanella-Béguelin, "Formal verification of smart contracts: Short paper," in *Proc. ACM Workshop Program. Lang. Anal. Security*, Oct. 2016, pp. 91–96.

[56] Y. Hirai. *Formal Verification of Deed Contract in Ethereum Name Service*. Accessed: May. 18, 2019. [Online]. Available: https://yoichihirai.com/deed.pdf

[57] S. Amani, M. Bégel, M. Bortin, and M. Staples, "Towards verifying ethereum smart contract bytecode in Isabelle/HOL," in *Proc. 7th ACM SIGPLAN Int. Conf. Certified Programs Proofs*, Jan. 2018, pp. 66–77.
[58] A. Danel, B. Bengamin, and B. Karthikeyan. *Pronounced F Star*. Accessed: May. 18, 2019. [Online]. Available: https://fstar-lang.org
[59] I. Grishchenko, M. Maffei, and C. Schneidewind, "A semantic framework for the security analysis of Ethereum smart contracts," in *Proc. Int. Conf. Princ. Secur. Trust*, Feb. 2018, pp. 243–269.
[60] Z. Yang and H. Lei, "Formal process virtual machine for smart contracts verification," *Comput. Res. Repository*, vol. 14, pp. 1–9, Aug. 2018. doi: 10.23940/ijpe.18.08.p9.17261734.
[61] Z. Yang and H. Lei, "Optimization of executable formal interpreters developed in higher-order logic theorem proving systems," *IEEE Access*, vol. 6, pp. 70331–70348, 2018. doi: 10.1109/ACCESS.2018.2880692.
[62] M. Matteo, S. Clara, G. Niklas, and J. Ilya. *EtherTrust-Trustworthy Smart Contracts*. Accessed: May. 18, 2019. [Online]. Available: https://www.netidee.at/ethertrust
[63] I. Grishchenko, M. Maffei, and C. Schneidewind, "Foundations and tools for the static analysis of Ethereum smart contracts," in *Proc. Int. Conf. Comput. Aided Verification*, Jul. 2018, pp. 57–78. doi: 10.1007/978-3-319-96145-3_4.
[64] J. Ellul and G. J. Pace, "Runtime verification of Ethereum smart contracts," in *Proc. 14th Eur. Dependable Comput. Conf.*, Sep. 2018, pp. 158–163. doi: 10.1109/EDCC.2018.00036.
[65] G. Bigi, A. Bracciali, G. Meacci, and E. Tuosto, "Validation of decentralised smart contracts through game theory and formal methods," in *Programming Languages with Applications to Biology and Security*. New York, NY, USA: Springer, Nov. 2015, pp. 142–161. doi: 10.1007/978-3-319-25527-9_11.
[66] X. Bai, Z. Cheng, Z. Duan, and K. Hu, "Formal modeling and verification of smart contracts," in *Proc. 7th Int. Conf. Softw. Comput. Appl.*, Feb. 2018, pp. 322–326. doi: 10.1145/3185089.3185138.
[67] T. Abdellatif and K.-L. Brousmiche, "Formal verification of smart contracts based on users and blockchain behaviors models," in *Proc. 9th IFIP Int. Conf. New Technol., Mobility Secur. (NTMS)*, Feb. 2018, pp. 1–5. doi: 10.1109/NTMS.2018.8328737.
[68] S. Fenech, G. J. Pace, and G. Schneider, "Automatic conflict detection on contracts," in *Proc. Int. Colloq. Theor. Aspects Comput.*, Aug. 2009, pp. 200–214. doi: 10.1007/978-3-642-03466-4_13.
[69] D. Gorin, S. Mera, and F. Schapachnik, "Model checking legal documents," in *Proc. Conf. Legal Knowl. Inf. Syst. JURIX*, Oct. 2010, pp. 151–154.
[70] C. Reitwiessner. *Dev Update: Formal Methods*. Accessed: May. 18, 2019. [Online]. Available: https://blog.ethereum.org/2016/09/01/formal-methods-roadmap/
[71] P. Rizzo. *formal Verification Push Ethereum Seeks Smart Contract Certainty*. Accessed: May. 18, 2019. [Online]. Available: http://www.coindesk.com/ethereum-formal-verification-smart-contracts/

JING LIU received the Ph.D. degree in computer architecture from the Institute of Computing Technology, Chinese Academy of Sciences, China, in 2011. He is currently an Associate Professor of computer science and technology with Inner Mongolia University. His major research interests include blockchain technology, software engineering, and formal methods. He has published more than 20 papers in international conferences and journals. He is a member of the IEEE.

ZHENTIAN LIU received the bachelor's degree in computer architecture from Shandong University, China, in 2017. She is currently pursuing the master's degree with Inner Mongolia University. Her research interests include formal methods and blockchain smart contracts.