Future Generation Computer Systems 105 (2020) 475–491

Contents lists available at ScienceDirect

Future Generation Computer Systems

journal homepage: www.elsevier.com/locate/fgcs

An overview on smart contracts: Challenges, advances and platforms

Zibin Zheng [a], Shaoan Xie [a], Hong-Ning Dai [b,*], Weili Chen [a], Xiangping Chen [a], Jian Weng [c], Muhammad Imran [d]

[a] School of Data and Computer Science, Sun Yat-sen University, China
[b] Faculty of Information Technology, Macau University of Science and Technology, Macao Special Administrative Region of China
[c] College of Information Science and Technology, Jinan University, Guangzhou, China
[d] College of Applied Computer Science, King Saud University, Riyadh, Saudi Arabia

ARTICLE INFO

Article history:
Received 20 June 2019
Received in revised form 5 December 2019
Accepted 12 December 2019
Available online 17 December 2019

Keywords:
Smart contract
Blockchain
Cryptocurrency
Decentralization

ABSTRACT

Smart contract technology is reshaping conventional industry and business processes. Being embedded in blockchains, smart contracts enable the contractual terms of an agreement to be enforced automatically without the intervention of a trusted third party. As a result, smart contracts can cut down administration and save services costs, improve the efficiency of business processes and reduce the risks. Although smart contracts are promising to drive the new wave of innovation in business processes, there are a number of challenges to be tackled. This paper presents a survey on smart contracts. We first introduce blockchains and smart contracts. We then present the challenges in smart contracts as well as recent technical advances. We also compare typical smart contract platforms and give a categorization of smart contract applications along with some representative examples.

1. Introduction

Blockchain technology has recently fueled extensive interests from both academia and industry. A blockchain is a distributed software system allowing transactions to be processed without the necessity of a trusted third party. As a result, business activities can be completed in an inexpensive and quick manner. Moreover, the immutability of blockchains also assures the distributed trust since it is nearly impossible to tamper any transactions stored in blockchains and all the historical transactions are auditable and traceable.

Blockchain technology is enabling *smart contracts* that were first proposed in 1990s by Nick Szabo [1]. In a smart contract, contract clauses written in computer programs will be automatically executed when predefined conditions are met. Smart contracts consisting of transactions are essentially stored, replicated and updated in distributed blockchains. In contrast, conventional contracts need to be completed by a trusted third party in a centralized manner consequently resulting in long execution time and extra cost. The integration of blockchain technology with smart contracts will make the dream of a "*peer-to-peer market*" come true.

Take a smart contract between a buyer and a supplier as an example. As shown in Fig. 1, a supplier first sends a product catalog to a buyer through the blockchain network. This catalog that includes product descriptions (such as property, quantity, price and availability) along with shipping and payment terms is stored and distributed in the blockchain so that a buyer can obtain the product information and verify the authenticity and reputation of the supplier at the same time. The buyer then submits the order with the specified quantity and payment date via the blockchain. This whole procedure forms a purchase contract (*i.e.*, *Contract 1*) enclosed in the blue box as shown in Fig. 1. It is worth mentioning that the whole procedure is completed between the buyer and the supplier without the intervention of a third party.

After *Contract 1* is done, the supplier will search for a carrier in the blockchain to complete the shipping phase. Like *Contract 1*, the carrier also publishes the shipping description (such as transportation fees, source, destination, capacity and shipping time) as well as shipping conditions and terms in the blockchain. If the supplier accepts the contract issued by the carrier, the products will be delivered to the carrier who will finally dispatch the products to the buyer. This whole procedure constructs *Contract 2* (enclosed in the pink box) as shown in Fig. 1. Similarly, the whole procedure of *Contract 2* is also conducted without the intervention of a third party.

In addition to automatic execution of *Contract 1* and *Contract 2*, the payment procedures (including the payment from the supplier to the carrier and that from the buyer to the supplier) are also completed automatically. For example, once the buyer confirms the reception of the products, the payment between the buyer and the supplier will be automatically triggered as the predefined condition is met. The financial settlement from the

* Corresponding author.
E-mail address: hndai@ieee.org (H.-N. Dai).

https://doi.org/10.1016/j.future.2019.12.019
0167-739X/

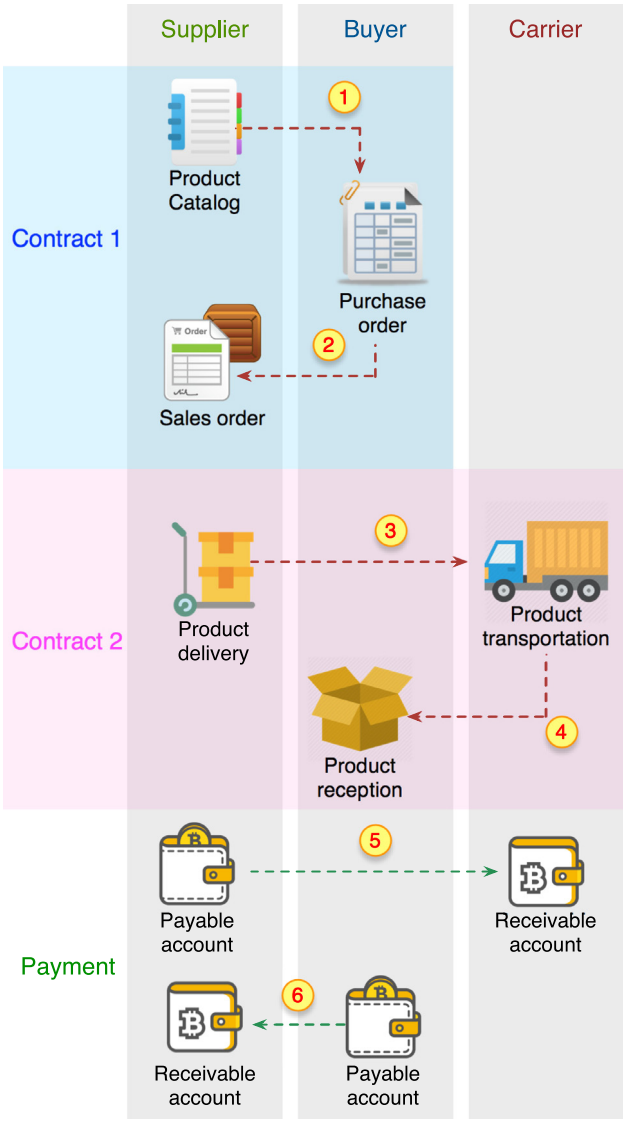

Fig. 1. An example of a smart contract between a buyer and a supplier.

Table 1
Acronym table.

Terms	Acronyms
Proof of Work	PoW
Proof of Stake	PoS
Practical Byzantine-Fault Tolerance	PBFT
Low Level Virtual Machine	LLVM
Convolutional Neural Network	CNN
Long Short Term Memory	LSTM
Ether	ETH
Bitcoin	BTC
Ethereum Virtual Machine	EVM
Unspent Transaction Output	UTXO
Internet of Things	IoT
Distributed Autonomous Corporation	DAC
Certificate Authority	CA
Delegated Proof of Stake	DPOS
WebAssembly	Wasm
Border Gateway Protocol	BGP

buyer to the supplier is conducted via crypto currencies (*e.g.*, Bitcoin or Ether.[1].). In contrast to conventional transactions, the whole process is done in a peer-to-peer manner without the intervention of third parties like banks. As a result, the turnaround time and transactional cost can be greatly saved.

In summary, smart contracts have the following advantages compared with conventional contracts:

- *Reducing risks.* Due to the immutability of blockchains, smart contracts cannot be arbitrarily altered once they are issued. Moreover, all the transactions that are stored and duplicated throughout the whole distributed blockchain system are traceable and auditable. As a result, malicious behaviors like financial frauds can be greatly mitigated.
- *Cutting down administration and service costs.* Block-chains assure the trust of the whole system by distributed consensus mechanisms without going through a central broker or a mediator. Smart contracts stored in blockchains can be automatically triggered in a decentralized way. Consequently, the administration and services costs due to the intervention from the third party can be significantly saved.
- *Improving the efficiency of business processes.* The elimination of the dependence on the intermediary can significantly improve the efficiency of business process. Take the aforementioned supply-chain procedure as an example. The financial settlement will be automatically completed in a peer-to-peer manner once the predefined condition is met (*e.g.*, the buyer confirms the reception of the products). As a result, the turnaround time can be significantly reduced.

Smart contracts are boosting a broad spectrum of applications ranging from industrial Internet of Things to financial services [2–11]. Although smart contracts have great potentials to reshape conventional business procedures, there are a number of challenges to be solved. For example, even if blockchains can assure a certain anonymity of the parties of the contract, the privacy of the whole contract execution may not be preserved since all the transactions are globally available. Moreover, it is challenging to ensure the correctness of smart contracts due to vulnerabilities of computer programs to the faults and failures.

There are some recent studies on smart contracts. For example, [12–14] present comprehensive surveys of blockchain technology and briefly introduce smart contracts. The work of [15] provides an in-depth survey on Ethereum smart contract programming vulnerabilities while [17] presents a detailed survey over verification methods on smart contract languages. The work of [16] reports authors' experiences in teaching smart contract programming and summarizes several typical types of mistakes made by students. Ref. [18] presents an empirical analysis on smart contract platforms. Recent studies [19,20] also collect some literature of smart contracts and present reviews while fail to discuss the challenges in this area. Moreover, the work of [21] presents a brief overview of smart contract platforms and architectures. However, most of existing papers fail to identify the rising challenges and give a comprehensive survey. For example, Ethereum can be used to conduct illegal business such as Ponzi schemes that were reported to defraud over 410,000 US dollars while few studies address this issue [22]. We summarize the differences between this paper and existing studies in Table 2.

The objective of this paper is to conduct a systematic overview of technical challenges in smart contracts enabled by blockchain technologies. Contributions of this paper are highlighted as following:

- Important research challenges in the life cycle of smart contracts are identified.
- Recent advances in addressing technical challenges are summarized.
- A detailed comparison of typical smart contract platforms is made.

[1] Commonly used acronyms in this paper are listed in Table 1

Table 2
Comparison with related work.

Research	Ethereum	Other platforms	Programming Languages	Other technical challenges	Technical advances	Rising challenges	Applications
[12–14]	✓	✗	✗	✗	✗	✗	✗
[15,16]	✓	✗	✓	✗	✗	✗	✗
[17]	✓	✓	✓	✗	✗	✗	✗
[18]	✓	✓	✓	✗	✗	✗	✓
[19–21]	✓	✓	✓	✓	✓	✗	✓
This paper	✓	✓	✓	✓	✓	✓	✓

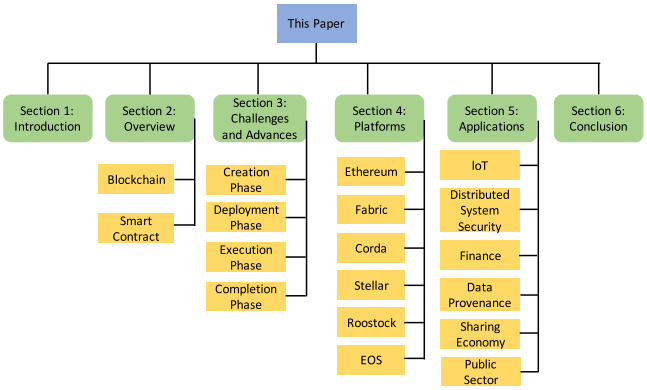

Fig. 2. Organization of this paper.

- Diverse smart contract applications are summarized.

Fig. 2 shows the organization of this paper. In particular, Section 2 gives a brief introduction to blockchains and smart contracts. Section 3 then summarizes research challenges in smart contracts as well as recent technical advances. Section 4 next compares typical smart contract development platforms. Section 5 categorizes typical smart contract applications. Finally, Section 6 concludes the paper.

2. Overview of blockchain and smart contract

Smart contracts are built upon blockchain technology ensuring the correct execution of the contracts. We first provide a brief introduction to blockchain technology in Section 2.1. We then give an overview on smart contracts in Section 2.2.

2.1. Blockchain

A blockchain can be regarded as a public ledger, in which all transactions cannot be falsified. Fig. 3 illustrates an example of a blockchain. A blockchain is a continuously-growing chain of blocks. When a new block is generated, all the nodes in the network will participate in validating the block. Once the block is validated, it will be appended to the blockchain.

To validate the trustfulness of blocks, consensus algorithms are developed. Consensus algorithms determine which node to store the next block and how the new appended block to be validated by other nodes. Representative consensus algorithms include proof of work (PoW) [23] and proof of stake (PoS) and practical byzantine-fault tolerance (PBFT) [24,25]. Consensus algorithms are usually done by users who first solve the puzzle (*i.e.*, PoW or PoS). These users are called *miners*. Each miner keeps a full copy of the blockchain. Different from PoW and PoS, PBFT requires a multi-round voting process to reach the consensus. The distributed consensus algorithms can ensure that transactions are done without the intervention of third parties like banks. As a result, the transaction costs can be saved. Moreover, users transact with their virtual addresses instead of real identities so that the privacy of users is also preserved.

In blockchain systems, it is possible that several nodes can successfully reach the consensus (*i.e.*, solving the puzzle) at the same time, consequently it can cause the bisected branches. To solve the disparity, a shorted side chain is desolated as shown in Fig. 3 while the longest chain is selected as the valid chain. This mechanism is effective since the longer chain is more tolerant to malicious attacks than the shorter chain in distributed systems.

In summary, blockchain technology has the key characteristics of decentralization, immutable, persistency and anonymity [26–28].

2.2. Smart contract

Smart contracts can be regarded as a great advance in blockchain technology [29]. In 1990s, a smart contract was proposed as a computerized transaction protocol that executes the contractual terms of an agreement [1]. Contractual clauses that are embedded in smart contracts will be enforced automatically when a certain condition is satisfied (*e.g.*, one party who breaches the contract will be punished automatically).

Blockchains are enabling smart contracts. Smart contracts are essentially implemented on top of blockchains. The approved contractual clauses are converted into executable computer programs. The logical connections between contractual clauses have also been preserved in the form of logical flows in programs (*e.g.*, the `if-else-if` statement). The execution of each contract statement is recorded as an immutable transaction stored in the blockchain. Smart contracts guarantee appropriate access control and contract enforcement. In particular, developers can assign access permission for each function in the contract. Once any condition in a smart contract is satisfied, the triggered statement will automatically execute the corresponding function in a predictable manner. For example, Alice and Bob agree on the penalty of violating the contract. If Bob breaches the contract, the corresponding penalty (as specified in the contract) will be automatically paid (deducted) from Bob's deposit.

The whole life cycle of smart contracts consists of four consecutive phases as illustrated in Fig. 4:

(1) *Creation* of smart contracts. Several involved parties first negotiate on the obligations, rights and prohibitions on contracts. After multiple rounds of discussions and negotiations, an agreement can reach. Lawyers or counselors will help parties to draft an initial contractual agreement. Software engineers then convert this agreement written in natural languages into a smart contract written in computer languages including declarative languages and logic-based rule languages [30]. Similar to the development of computer software, the procedure of the smart contract conversion is composed of design, implementation and validation (*i.e.*, testing). It is worth mentioning that the creation of smart contracts is an iterative process involving with multiple rounds of negotiations and iterations. Meanwhile, it is also involved with multiple parties, such as stakeholders, lawyers and software engineers.

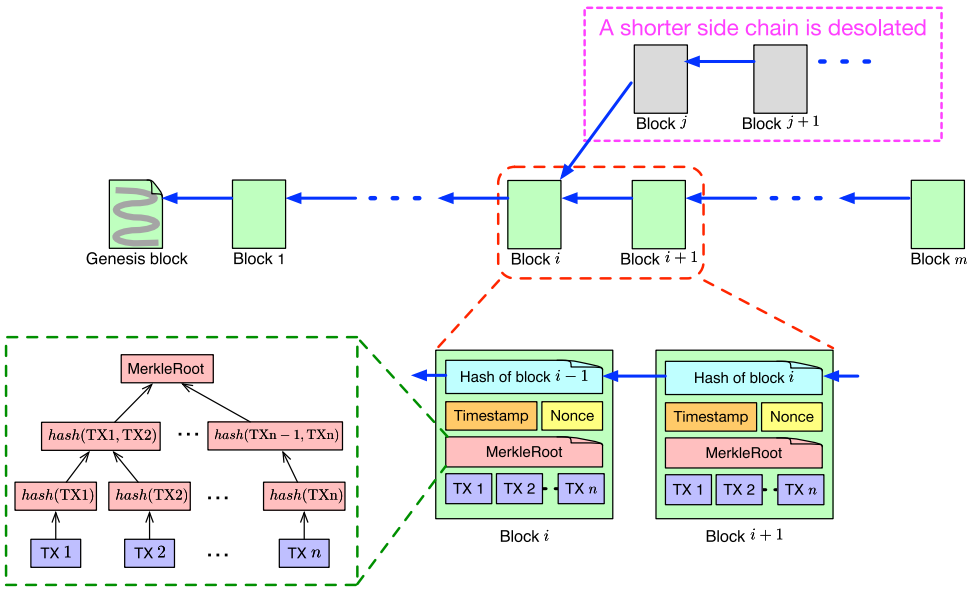

Fig. 3. A blockchain consists of a sequence of blocks, each of which contains an inverse hash pointing back to its parent block. Meanwhile, there are a number of transactions stored inside a block.

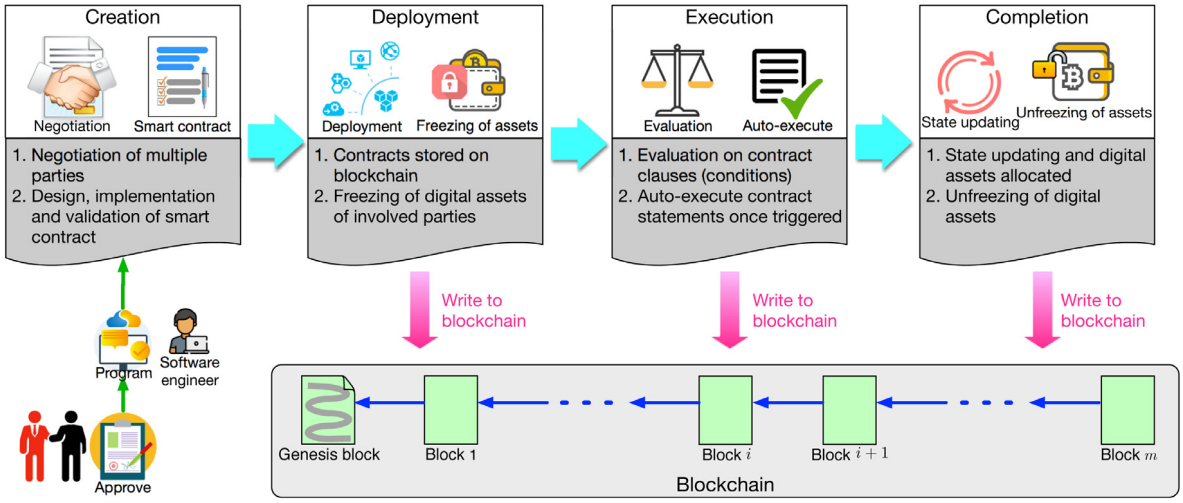

Fig. 4. A life cycle of a smart contract consists of four major phases: Creation, Deployment, Execution and Completion.

(2) *Deployment* of smart contracts. The validated smart contracts can then be deployed to platforms on top of blockchains. Contracts stored on the blockchains cannot be modified due to the immutability of block-chains. Any emendation requires the creation of a new contract. Once *smart* contracts are deployed on blockchains, all the parties can access the contracts through the blockchains. Moreover, digital assets of both involved parties in the smart contract are locked via freezing the corresponding digital wallets [31]. For example, the coin transfers (either incoming or outgoing) on the wallets relevant to the contract are blocked. Meanwhile, the parties can be identified by their digital wallets.

(3) *Execution* of smart contracts. After the deployment of smart contracts, the contractual clauses have been monitored and evaluated. Once the contractual conditions reach (*e.g.*, product reception), the contractual procedures (or functions) will be automatically executed. It is worth noting that a smart contract consists of a number of declarative statements with logical connections. When a condition is triggered, the corresponding statement will be automatically executed, consequently a transaction being executed and validated by miners in the blockchains [32]. The committed transactions and the updated states have been stored on the blockchains thereafter.

(4) *Completion* of smart contracts. After a smart contract has been executed, new states of all involved parties are updated. Accordingly, the transactions during the execution of the smart contracts as well as the updated states are stored in blockchains. Meanwhile, the digital assets have been transferred from one party to another party (*e.g.*, money transfer from the buyer to the supplier). Consequently, digital assets of involved parties have been unlocked. The smart contract then has completed the whole life cycle.

It is worth mentioning that during deployment, execution and completion of a smart contract, a sequence of transactions has been executed (each corresponding to a statement in the smart contract) and stored in the blockchain. Therefore, all these three phases need to write data to the blockchain as shown in Fig. 4.

Table 3
Summary of challenges and advances in smart contracts.

Phases	Challenges	Advances
Creation	1) Readability	• Recover source code [33] • Human readable code [34], [35] • Human readable execution [36], [37]
	2) Functional issues	• Re-entrancy [38], [39], [40] • Block randomness [41], [42], [43] • Overcharging [44], [45]
Deployment	1) Contract correctness	• Bytecode analysis [46], [47], [48], [49], [50], [51], [52], [53] • Source code analysis [54], [55], [56], [57] • Machine learning based analysis [58], [59], [60]
	2) Dynamic control flow	• Graph based analysis [61], [62] • Path-searching [63] • Execution environment [64]
Execution	1) Trustworthy oracle	• Third-party involved [65] • Decentralized [66], [67]
	2) Transaction-ordering dependence	• Sequential execution [68] • Predefining contract [69]
	3) Execution efficiency	• Execution serialization [70], [71], [72] • Inspection of contract [73]
Completion	1) Privacy and Security	• Privacy [74], [75] • Security [76]
	2) Scam	• Ponzi scheme [77] [22], [78] • Honeypot [79]

3. Challenges and advances of smart contract

Although a smart contract is a promising technology, there are still a number of challenges to be tackled. We categorize these major challenges into four types according to four phases of the life cycle of smart contracts. Meanwhile, we also give an overview on recent advances in solving these challenges. Table 3 summarizes the challenges and recent advances.

3.1. Creation challenges

Contract creation is an important step to implement smart contracts. Users have to code their own contracts and then deploy them in various blockchain platforms (to be introduced in Section 4). Since blockchains are essentially immutable, blockchain-based smart contracts also cannot be modified after being deployed. As a result, developers need to carefully address the following problems.

3.1.1. Readability

Most of smart contracts are written in programming languages such as Solidity, Go, Kotlin and Java (to be described in Section 4). Then source codes will be compiled and executed. Therefore, in different time periods, programs have different forms of codes. How to make programs readable in each form remains a big challenge.

Recent advances for readability challenge

- *Recover source code*: It is shown in [33] that more than 77% smart contracts have not released public source codes, all of which are involved with over $3 billion US dollars. Unavailability of source code makes smart contract be opaque to the official auditors. To address this issue, [33] proposed a reverse engineering tool (namely Erays) to analyze compiled smart contracts. This reverse engineering tool is able to convert hex encoded contract into a human readable pseudo codes.
- *Human readable code*: Frantz and Nowostawski [34] proposed a semi-automated translation system that can convert human-readable contract representations into computational programs. Essentially, this semi-automated translation system has been implemented according to the concept from institutional analysis [80]. In particular, institution specifications can be decomposed into different components such as `attributes`, `deontic`, `aim`, `conditions` and `or else`. These components are then mapped into the corresponding blocks written in programming languages. For example, the `attributes` can be converted into `structs` in Solidity. Since most of smart contract programming languages are object-oriented languages, [35] argues that declarative language backed by a logic programming computational model might be more suitable for smart contract. For example, the authors claim that Prolog (a type of logic language) does not requires compilation so it also avoids the inspection on smart contract compilation.
- *Human readable execution*: Although many platforms attempt to provide smart contract developers with high level languages, these smart contracts will then be compiled into other forms, e.g., bytecode in *Ethereum Virtual Machine* (EVM). In most of cases, two parties in the transaction need to understand the contracts at the level that it has been stored and executed on blockchain. Ref. [36] proposed an intermediate level language named IELE to solve this challenge. IELE has a similar syntax to Low Level Virtual Machine (LLVM) [37] so as to provide compilers with high-level information during the compile time, link time, run time, and idle time.

3.1.2. Functional issues

There are a number of functional issues with incumbent smart contract platforms. We present several representative challenges: (1) *Re-entrancy* means that the interrupted function can be safely recalled again. Malicious users may exploit this vulnerability to steal digital currency as indicated in [81]. (2) *Block randomness.* Some smart contract applications such as lotteries and betting pools may require randomness of generated blocks. This can be achieved by generating pseudo-random numbers in a block timestamp or nonce. However, some malicious miners may fabricate some blocks to deviate from the outcome of the pseudo-random generator. In this way, attackers can control the probability distribution of the outcomes as shown in [82]. (3) *Overcharging.* It is shown in recent work [44] that smart contracts can be overcharged due to the under-optimization of smart contracts.

These overcharged patterns have the features like dead code, expensive operations in loops consisting of repeated computations.

Recent advances for functional issues

- *Re-entrancy*: Recently, several proposals attempt to solve some of the above challenges. Obsidian [38] was proposed to address re-entrancy attacks and money leakage problems. In particular, Obsidian exploits *named* states to enable consistency checking on state transitions and verification so that re-entrancy vulnerability can be mended. Moreover, a data flow analytical method was proposed to prevent the illegal digital currency stealing from the leakage. Ref. [39] proposed to eliminate re-entrancy vulnerabilities by prohibiting the nesting calling among functions in the contract. Liu et al. [40] proposed to perform the fuzz testing on smart contracts by iteratively generating random but diverse transactions to detect re-entrancy bugs.
- *Block randness*: Blockchain is regarded as a promising technology to generate public and unpredictable random values. However, the random output might not be so random as people expect. Miners could control the block generation and release the block until they find it profitable. To address this issue, [41] proposed to use the delay-function to generate randomness. It means that the random value will be only be known to others after a short time period since its generation. In this way, the blockchain moves on and the miners could not withhold their blocks to profit. But delay functions are not suitable for smart contracts as most of they require instance verification. To this end, [42] proposed the Sloth function to allow faster verification. Based on [42,43] proposed a multi-round protocol to verify delay functions using a refereed delegation model. It reduces the cost of verifying the output from $30 to $0.4.
- *Overcharging*: Besides from caring the efficiency of their programs, developers of smart contract also need to pay attention to their execution costs. Ref. [44] reported that over 90% of real smart contracts suffer from gas-costly patterns in Ethereum. Chen et al. [45] proposed GasReducer, a tool used to detect gas-costly patterns. GasReducer can replace under-optimized bytecode with efficient bytecode.

3.2. Deployment challenges

After creation, smart contracts will be deployed on blockchain platforms. But smart contracts need to be checked carefully to avoid potential bugs. Furthermore, smart contract developers need to be aware of the contract's interaction patterns to mitigate potential losses due to the malicious behaviors (such as frauds and attacks [83]). We next describe the challenges as well as advances in smart contract deployment.

3.2.1. Contract correctness

Once smart contracts have been deployed on blockchains, it is nearly impossible to make any revisions. Therefore, it is of vital importance to evaluate the correctness of smart contracts before the formal deployment. However, it is challenging to verify the correctness of smart contracts due to the complexity of modeling smart contracts.

Recent advances for contract correctness

- *Bytecode analysis*: Bytecode level analysis only requires the compiled bytecodes of smart contracts, which are much easier to obtain. How to utilize these bytecode to detect security threats has become a hot research topic. In particular, OYENTE was proposed in [46] to identify potential security bugs including mishandled exceptions and timestamp-dependent problems. Based on the control graph generated by OYENTE, [47] produces the rule-based representations for high level bytecode analysis. Meanwhile, Knecht and Stiller [48] proposed a smart contract deployment and management platform (SmartDEMAP) to address the trust problem during the contract development and deployment. Moreover, other code quality control tools such as automated bug-finders can also be equipped with SmartDEMAP. In this manner, smart contracts can be deployed only after the trustful conditions are fulfilled. Ref. [49] proposed MadMax to predict gas-focus vulnerabilities in Ethereum smart contracts. The combination of control-flow-analysis-based decompiler and declarative program-structure queries enables the method detecting the vulnerabilities in high precision. Meanwhile [84] symbolically analyzed the contract dependency graph to extract precise semantic information from the code. Then, it checks compliance and violation patterns that capture sufficient conditions to prove if a property holds or not. Furthermore, [50] proposed a method to search for certain critical paths in the control flow graph of a smart contract and identify paths that may lead to a critical instruction, where the arguments of instructions can be controlled by an attacker. Ref. [51] proposed Vandal, a tool which firstly converts low level bytecode into register transfer language that was then translated into logic semantic relations. In addition, [52] proposed Gigahorse, a tool which is able to decompile smart contract bytecode to high level 3-address code representation. The new intermediate representation of smart contracts makes the implicit data and control flow dependencies of the EVM bytecode be explicit. Amani et al. [53] reconstructed bytecode sequences into blocks of straight-line code and created a program logic to identify the security vulnerability of the contract.
- *Source code analysis*: Compared with bytecode level analysis, source code analysis requires the availability of smart contract source codes. Although the source code analysis contains more information, it also requires highly-precise analysis. There are a number of studies on source code analysis of smart contracts. In particular, a formal verification method was proposed in [54] to analyze and verify both the runtime safety and the functional correctness of smart contracts (*e.g.*, Ethereum contracts). This method first translates smart contracts into codes written in F* [55], which is a functional programming language mainly used for program verification. This translation can be used to detect abnormal patterns like *stack overflow* (*i.e.*, exceeding the stack limit). Meanwhile, [56] proposed Zeus to verify the correctness of smart contracts. Zeus firstly translates the contracts and policy specification into low-level intermediate representation and feeds the encoded representation into constrained horn clauses [57] to ascertain the safety of the smart contract.
- *Machine learning based analysis*: Recently, machine learning-based methods have been proposed to obtain a better representation for detecting vulnerabilities in smart contracts. In particular, [58] proposed a novel semantic-aware security auditing technique called the *S*-gram scheme for Ethereum. The *S*-gram scheme that combines the *N*-gram language modeling and static semantic labeling can be used to predict potential vulnerabilities by identifying irregular token sequences and optimize existing in-depth analyzers. Meanwhile, the work of [59] translates the bytecode of smart contract into RGB color that was transformed into images. The images were fed into a convolutional neural network (CNN) to extract more meaningful features. Moreover, [60] applied Long Short Term Memory (LSTM) to analyze the security threats of smart contracts at an opcode level.

3.2.2. Dynamic control flow

Despite the fact that the deployed smart contracts are immutable, the control flow of smart contracts is not guaranteed to be immutable. In particular, a smart contract can interact with other contracts (*e.g.*, transferring funds to the contract or creating a new contract). The control flow of smart contract needs to be designed carefully when developing the contract. The interaction of smart contracts can result in an increased number of interconnected contracts over time. Therefore, how to predict the contract behaviors becomes challenging. In addition, most of existing methods pay attention to the detection of potential dynamic control flow problems in programs while the reliability of the execution environment is not always ensured. Therefore, it is also significant to check whether the execution environment is reliable.

Recent advances for dynamic control flow

- *Graph based analysis*: Charlier et al. [61] proposed a multi-dimensional approach to predict interactions among smart contracts. In particular, this approach integrates stochastic processes and tensors to reproduce existing interactions, consequently predicting future contract interactions. Furthermore, the work in [62] presents a heuristic indicator of control flow immutability. In particular, this approach was evaluated on a call graph of all smart contracts on Ethereum. Through analyzing the call graph, it is shown that two smart contracts (out of five) require a trust in at least one third party.
- *Path-searching*: Nikolić et al. [63] proposed a method namely MAIAN to detect vulnerabilities across a long sequence of invocations of a contract. MAIAN employs inter-procedural symbolic analysis and concrete validator for exhibiting real exploits. It searches the spaces of all execution paths in a trace with depth-first search (DFS) and checks whether the contract triggers property violation. Different from above graph-based methods, MAIAN is designed to identify either locking funds indefinitely, leakage to arbitrary users or being killed by anyone. Therefore, it does not need to model the interactions among smart contracts.
- *Execution environment*: EVMFuzz [64] was proposed to detect vulnerabilities of the execution environment of smart contract. EVMFuzz continuously generate seed contracts for different EVM executions, so as to find as many inconsistencies among execution results as possible. This method can eventually discover vulnerabilities with cross-referencing outputs.

3.3. Execution challenges

Execution phase is crucial to smart contracts as it determines the final state of smart contracts. There are a number of issues to be addressed during the execution of smart contracts.

3.3.1. Trustworthy oracle

Smart contracts cannot work without real-world information. For example, an Eurobet (*i.e.*, a soccer betting smart contract) needs to know the result of European Cup. However, a smart contract is designed to run in a sandbox isolating from the outside network. In a smart contract, an *oracle* plays a role of an agent who finds and verifies real-world occurrences and forwards this information to the smart contract. Thus, how to determine a trustworthy *oracle* becomes a challenge.

Recent advances for trustworthy oracle

- *Third-party involved*: Town Crier (TC) [65] was proposed to address this challenge. In particular, TC scrapes data from reliable web sites and feeds those data to smart contracts. TC feeds the data in the form of datagram that is accompanied with the specific data-source web site and a concrete time frame. Meanwhile, TC executes its core functionality in a Software Guard Extension (SGE) enclave that protects TC from attacks of malicious behaviors.
- *Decentralized*: Ref. [66] proposed a decentralized oracle named ASTRAEA, which is based on a voting game among stake-holders. In particular, voters place a reasonable amount of stakes to vote the random proposition selected from the system. Once the weighted sum of votes matches the vote from a voter, the voter will be rewarded, otherwise, the voter will be penalized. Meanwhile, [67] proposed a smart contract based solution for selecting trustworthy oracles. A reputation contract is used to record each oracle-service-provider's reputation according to its previous performance. Then an aggregating contract will calculate the final results of a query from users and finalize the result.

3.3.2. Transaction-ordering dependence

Users send transactions to invoke functions in a smart contract while miners pack the transactions into blocks. However, the order of transactions is not deterministic due to the uncertainty of the bisected blockchain branches [26]. This uncertainty can cause inconsistency of order-dependent transactions. For example, there is a contract containing variant x. Alice sends a transaction to increase x by 1 while Bob sends a transaction to multiply x by 10. Due to uncertainty of the transaction order, the final outcomes on variant x can either be $x + 1$ or $x \times 10$. It is worth mentioning that this inconsistency has been well solved in conventional database management systems (DBMS) [85] while it is challenging to solve it in smart contracts as far as we know.

Recent advances for transaction-ordering dependence

- *Sequential execution*: Ref. [68] introduced a design pattern of smart contract-transaction counter. Transaction counter expects a transition number in each function as a parameter and ensures the number be increased by one after each function execution. Through analyzing the transition number, the inconsistency problem is solved.
- *Predefining contract*: To avoid such anomaly, [69] proposed to write smart contracts instead of transactions. For example, if Alice wants to increase the value of x after Bob's operation, she writes a `IncreaseIfMultiplied()` function, which avoids the situation where Alice's operation executes prior to Bob's.

3.3.3. Execution efficiency

Smart contracts are serially executed by miners. In other words, a miner will not execute another contract until the current contract is completed. The execution serialization essentially limits the system performance. However, it is challenging to execute smart contracts concurrently due to the shared data between multiple smart contracts. In the meantime, how to inspect the contract data without prescribed interface is also important to improving the smart contract execution efficiency as it removes the need to redeploy a new contract.

Recent advances for execution efficiency

- *Execution serialization*: To fill this gap, Dickerson et al. [70] proposed an approach based on Software Transactional Memory to allow miners or validators to execute contracts

in parallel. The main idea of this approach is to treat each invocation of a smart contract as a speculative atomic action. In this way, conflicts happened during the parallel executions can be rolled back easily. Furthermore, the work in [71] investigated smart contracts in a concurrent perspective. In particular, concurrency issues such as atomicity, interference, synchronization, and resource ownership have been well studied in this paper. Ref. [72] proposed to use optimistic Software Transactional Memory Systems to help improve the execution efficiency of smart contracts. While executing contract transactions concurrently using multi-threading, the miner also stores a block graph of transactions into the block. Then the validators re-execute the smart contract concurrently with the given block graph. If the result is consistent, the block will be appended into the blockchain.

- *Inspection of contract*: After deployment, the contract content cannot be modified. What can developers do if they are asked to observe some values that are not described in their initial requirements? A straightforward solution is to amend the smart contract and re-deploy it. However, the redeployment of smart contracts may cause additional costs. Ref. [73] proposed to exploit the memory layout *reification* to decompile the binary structure of a compiled contract. Meanwhile, the work of [73] proposed the decompilation capabilities encapsulated in mirrors [86], through which the method can introspect the current state of a smart contract instance without redeploying it.

3.4. Completion challenges

After the execution of smart contract, the modification to the states in the system will be packed as a transaction and broadcasted to each node. However, the proliferation of smart contracts brings additional concerns.

3.4.1. Privacy and security

Most current smart contract and blockchain platforms lack of privacy-preserving mechanisms, especially for transactional privacy. In particular, the transaction records (*i.e.*, the sequence of operations) are disseminated throughout the whole blockchain networks. Consequently, all the transactions are visible to everyone in the networks. Although some blockchain systems utilize pseudonymous public keys to improve the anonymity of the transactions, most transaction data (such as balances) are still publicly visible. As shown in [87], it is possible to obtain useful information from the transaction data based on the transactional graph analysis. smart contract systems also have their inherent software vulnerabilities, which are susceptible to malicious attacks. In addition, smart contracts run on top of blockchain systems which are also suffering from system vulnerability. For example, it is reported in [88] that attackers exploited Border Gateway Protocol (BGP) routing scheme to intercept messages in blockchains. It can cause high delay of message broadcasting and also hijack the traffic of a subset of nodes, thereby stealing digital currency.

Recent advances for privacy and security

- *Privacy*: To address the privacy concerns of smart contracts, Kosba et al. [74] proposed Hawk — a decentralized smart contract system to establish privacy-preserved smart contracts. In particular, the Hawk compiler will compile a contract into a cryptographic protocol automatically. The compiled Hawk program contains two major parts: a private portion used to execute the major function and a public portion used to protect users. Hawk will encrypt the transaction information (*e.g.*, transaction balance) and verify the correctness of transactions via using zero-knowledge proofs (*i.e.*, without viewing the content of the transactions). The anonymity of the parties in smart contracts can be ensured while the secrecy of contract execution may not be fulfilled. Enigma [75] offers a solution to the secrecy of smart contract execution. Advanced cartographic algorithms are used in Enigma to support zero-knowledge proofs. Moreover, Enigma distributes blockchain data in different nodes unlike traditional blockchain redundant schemes (*i.e.*, every node saves a copy of all transactions).
- *Security*: There are some efforts in solving the security concerns. For example, the recent work of [76] proposes a secure relaying-network for blockchains, namely SABRE. In particular, SABRE adjusts the inter-domain routing policies for BGP routing scheme. It can protect the link between clients and relays via placing relays appropriately. Meanwhile, SABRE also adopts the co-design of hardware and software via software defined networking (SDN) to reduce the traffic burden at relays. Experimental results demonstrate the effectiveness against BGP routing attacks.

```
contract Multiplicator {
    ...
    function multiplicate(address receiver) payable {

    if (msg.value >= this.balance)
     {receiver.transfer(this.balance+msg.value);}

}
```

Fig. 5. Example of smart contract honeypot [79].

3.4.2. Scams

As a new technology, blockchain and smart contracts are vulnerable to malicious attacks initiated by scams. The detection of scams is of great importance especially for contract users since it enables them to terminate their investments at an early phase to avoid the unnecessary loss.

Recent advances for readability challenge

- *Ponzi scheme*: Ponzi scheme is a classical fraud which promises high return rates with little risk to investors. It pays the older investors with new investors' funds. But if there is no enough circulating money, the scheme unravels those posteriors who consequently lose their money. The recent work of [77] conducted a systematic study over the Ponzi schemes on Ethereum. In particular, 16,082,269 transactions were collected from July, 2015 to May, 2017. It was found that 17,777 transactions were related to Ponzi schemes, which had already collected over 410,000 US dollars within only two years. Chen et al. [22] proposed a method to extract features from both accounts and the operation codes to identify Ponzi schemes on Ethereum. Meanwhile, the work of [78] proposed a novel approach to detect and quantify Ponzi schemes on Bitcoin. In particular, to address the difficulty of identifying Ponzi schemes as they often use multiple addresses, a clustering method [78] was proposed to identify the addresses. They found that 19 out of 32 Ponzi schemes use more than one addresses.
- *Honeypot*: Smart contract Honeypot implies that the vulnerable-looking contracts contain hidden traps. Take Fig. 5 as an example. At the first glance, a naive user may believe that the contract will automatically return the total amount

of current balance plus extra money after he or she sends the money to this contract. However, the balance will increase prior to the function execution and the condition of `if (msg.value >= this.balance)` will never be satisfied. The work of [79] developed a taxonomy of honeypot techniques and use symbolic execution and heuristics to detect honeypots in smart contracts. In addition, [79] found that the honeypot contracts have made over $90,000 profit for creators.

4. Smart contract development platforms

Recently, smart contracts have been developed on block-chain-based platforms. These platforms provide developers with simple interfaces to build smart contract applications. Among a number of incumbent blockchain platforms, many of them can support smart contracts. In this paper, we introduce 5 most representative smart contract platforms: Ethereum [89], Hyperledger Fabric [90], Corda [91], Stellar [92], Rootstock [93] in Section 4.1. We choose them mainly due the popularity in developing community and technical maturity as implied in [94]. We next summarize the common features of them in Section 4.2. Finally, we give an example of developing a smart contract in Section 4.3.

4.1. Representative platforms

4.1.1. Ethereum

Ethereum is a decentralized platform that can execute smart contracts. In contrast to Bitcoin's *Turing-incomplete* script system, Ethereum has developed *Turing-complete* languages such as Solidity,[2] Serpent,[3] Low-level Lisp-like Language (LLL)[4] and Mutan[5] to support general user applications beyond cyptocurrency applications. Ethereum compiles smart contracts written by Solidity, Serpent, LLL and Mutan into machine codes, which will then be loaded to EVM and run. Meanwhile, Ethereum adopts the account-based data model, in which each participant is identified by its digital wallet.

Similar to Bitcoin, Ethereum adopts PoW as the consensus algorithm, which is also computational intensive. To compensate the cost of solving puzzles done by miners, *Ether* (ETH) instead of coins (BTC) in Bitcoin is used. Essentially, *gas* serves as an internal price for executing a transaction to overcome the unstable value of ETH. Informally, the total cost of a transaction can be calculated by $\texttt{gas\_limit} \times \texttt{gas\_price}$, where gas_limit denotes the maximum amount of gas to be used to generate a block and gas_price is the cost of a unit of gas (in ETH). Users can pay different amounts of gas to let their transactions be confirmed earlier or later (*i.e.*, large amount of gas resulting in the fast confirmation). Since PoW is computationally intensive, it can waste a number of electricity for meaningless block mining tasks. It is expected if the mining process is used for meaningful events, such as help solving mathematical puzzles and conducting machine learning tasks.

4.1.2. Hyperledger fabric

Hyperledger Fabric is also a distributed ledger platform for running smart contracts [90]. Different from Ethereum who runs smart contacts in virtual machines (*i.e.*, EVM), Hyperledger adopts Docker container to execute the code. In contrast to virtual machines (VMs), containers can support smart contract applications with lower overhead while sacrificing the isolation (*i.e.*, applications in one container are running on top of one operating system). Instead of developing smart contract languages of Etherum, Fabric supports conventional high-level programming languages such as Java and Go (*aka* Golang). Similarly, Fabric is also Turing complete. Fabric adopts *key–value* pair as the data model.

As Fabric is designed to support general enterprise applications, the Fabric blockchain-network is permissioned (private or consortium). Users have to be authorized to join the network by certificate authorities (CAs). Since there are different roles in the network, multiple types of CAs coexist. For instance, the *enrollment certificate authority* (ECA) allows users to register with blockchains. Once the user has registered, he/she has to request transaction certificates from *transaction certificate authority* (TCA). The consensus can be easily reached within the permissioned blockchain-network. Fabric exploits PBFT which requires the multi-round voting among authenticated users. PBFT relies on multi-round communications among nodes, which can cause time delay. More efficient consensus algorithms should be developed to resolve this problem.

4.1.3. Corda

In contrast to diverse applications of Ethereum, Corda [91] is specialized for digital-currency applications. It serves as a distributed-ledger platform for saving and processing historical digital-asset records. Corda adopts high-level programming languages such as Java and Kotlin,[6] which are running on top of Java Virtual Machine (JVM). Meanwhile, Corda is Turing incomplete to support the verifiability. Moreover, the data model in Corda is the transaction-based model.

Corda typically supports private platforms, in which enterprises establish an authored network to exchange digital-assets in the private manner. In private blockchain platforms, the consensus can easily reach. Corda adopts Raft [95] as the consensus algorithm. The consensus in Raft can be achieved by selecting a leader, log replication and safety assurance. Instead of globally broadcasting in blockchains, Corda uses the point-to-point messaging mechanism. Users have to specify the message receivers and the detailed information to be sent.

4.1.4. Stellar

Similar to Corda, Stellar [92] is a specialized platform for digital-currency applications. Compared with Ethereum, Stellar is simpler and more accessible. Meanwhile, Stellar can support a diversity of languages such as Python, JavaScript, Golang and PHP. However, Stellar contracts are not Turing complete. Similar to Fabric, Stellar executes program codes on top of Docker containers, consequently reducing the overhead. For example, the execution cost of one transaction at Stellar is only ∼$0.0000002, which can almost be ignored. Moreover, the execution time for one transaction in Stellar is about 5 s on average in contrast to 3.5 min in Ethereum. Therefore, Stellar is an ideal platform for digital-currency applications. Like Ethereum, Stellar adopts the account-based model as the data model. Stellar developed its own consensus algorithm — Stellar Consensus Protocol (SCP) [92]. Since Stellar is permissioned, the consensus can be easily reached via SCP.

4.1.5. Rootstock

Rootstock (RSK) [93] runs on top of Bitcoin while supporting faster execution of transactions. For example, RSK can confirm the executed transaction within 20 s. Meanwhile, RSK is compatible with Ethereum (*e.g.*, adopting Solidity to implement contracts).

[2] https://solidity.readthedocs.io/en/develop/.
[3] https://github.com/ethereum/wiki/wiki/Serpent.
[4] https://lll-docs.readthedocs.io/en/latest/lll_introduction.html.
[5] https://github.com/ethereum/go-ethereum/wiki.
[6] https://kotlinlang.org/.

Table 4
Comparison of Smart Contract Platforms.

	Ethereum	Fabric	Corda	Stellar	Rootstock	EOS
Execution environment	EVM	Docker	JVM	Docker	VM	WebAssembly
Language	Solidity, Serpent, LLL, Mutan	Java, Golang	Java, Kotlin	Python, JavaScript, Golang and PHP, etc.	Solidity	C++
Turing Completeness	Turing complete	Turing complete	Turing incomplete	Turing incomplete	Turing complete	Turing complete
Data model	Account-based	Key–value pair	Transaction-based	Account-based	Account-based	Account-based
Consensus	PoW	PBFT	Raft	Stellar Consensus Protocol (SCP)	PoW	BFT-DPOS
Permission	Public	Private	Private	Consortium	Public	Public
Application	General	General	Digital currency	Digital currency	Digital currency	General

RSK contracts are also Turing complete.[7] Furthermore, RSK developed its own virtual machines to run smart contracts. The data model of RSK is also account-based while RSK is a public blockchain system. RSK developed its consensus scheme based on PoW while it adopts lightweight implementation consequently reducing the overhead. Like Corda and Stellar, RSK was proposed to mainly support digital-currency applications. RSK has a merit, i.e., much safer than those systems independent of blockchains since it runs on top of Bitcoin. However, it can cause extra burden on Bitcoin blockchain. How to resolve this problem is crucial to RSK.

4.1.6. EOS

EOS [96] is designed to enable the scalability of decentralized applications. Instead of using one type of consensus algorithms only, EOS combines Byzantine Fault Tolerance (BFT) and Delegated Proof of Stake (DPOS), thereby obtaining the advantages of both consensus algorithms. At each round, delegates (*i.e.*, block producers) will be selected by stake holders to produce a new block and BFT will proceed among those selected delegates to make the block irreversible. Similar to Bitcoin, EOS is also account-based but it also allows all accounts to be referenced by human readable names. Instead of customizing a virtual machine for code execution like Ethereum, EOS chooses to use WebAssembly (Wasm) so that it is possible to write a smart contract in various languages as long as it can be compiled into Wasm (*e.g.*, EOS supports C++).

4.2. Comparison of smart contract platforms

Table 4 compares Ethereum, Fabric, Corda, Stella, Rootstock (RSK) and EOS from the following aspects such as execution environment, supporting language, Turing completeness, data model, consensus protocols, permission and application. We next summarize the main characteristics of these representative smart contract platforms as follows.

- *Execution Environment*. Contracts in Ethereum are executed in EVM. Similarly, Corda and Rootstock adopt JVM and RSK virtual machines, respectively. In contrast, Fabric and Stellar run smart contracts on top of Docker containers, consequently reducing the overhead while sacrificing the isolation of applications. EOS chooses to use Wasm to support more smart contract languages.
- *Supported Languages*. Ethereum supports Solidity, Serpent and Mutan, which are specially designed for Ethereum. Fabric currently supports Java and Golang while Corda adopts Java and Kotlin. Stellar can support a diversity of languages such as Python, Javascript, Golang and PHP. To be compatible with Ethereum, RSK adopts Solidity as the contract language while EOS currently only supports C++.
- *Turing completeness*. Smart contracts on Ethereum, EOS, Fabric and RSK are all Turing complete while Corda and Stellar are Turing incomplete. Turing-complete contracts are typically more expressive than Turing-incomplete contracts. However, the Turing completeness also brings the potential software bugs being susceptible to malicious attacks (as illustrated in Section 3).
- *Data Model*. Corda adopts the unspent transaction output (UTXO) model as Bitcoin does. In UTXO model, each payment has to specify the previous unspent transaction as the input. Then the specified transaction becomes *spent*. The changes will be made on new unspent transactions. Ethereum, Stellar, EOS and RSK adopt account-based model, in which the balance of an address is recorded directly instead of calculating all the unspent transaction amounts. Fabric exploits key–value model, in which data is represented in key–value pairs stored in blockchains.
- *Consensus Algorithms*. Ethereum and RSK adopt PoW, in which the validation of the trustfulness of a block is equivalent to the solution of a computationally difficult problem (*i.e.*, a puzzle). PoW consensus algorithms are typically computational-intensive. Fabric chooses to PBFT consensus algorithm [24], in which several rounds of voting among authenticated nodes are taken to reach the consensus. Therefore, PBFT is network-intensive. In contrast, Corda adopts a simple consensus algorithm namely Raft to achieve the consensus between different sectors at the level of individual deals instead of a global system. Similarly, Stellar develops a simply consensus algorithm named SCP to reach the consensus. EOS uses the combination of BFT and DPOS.
- *Permission*. Ethereum, EOS and RSK are public (*i.e.* permissionless) smart contract platforms and each user can arbitrarily join the network while Corda and Hyperledger are private platforms only allowing authenticated users to access. Stellar sitting between public and private blockchains is a consortium blockchain across different enterprise sectors (or organizations).
- *Applications of smart contract*. Unlike Corda, Stellar and RSK only support digital-currency while Ethereum and Fabric cater for a wider diversity of applications ranging from digital currency, digital-asset management, capital investment,

[7] Bitcoin is not Turing complete while Rootstock is Turing complete.

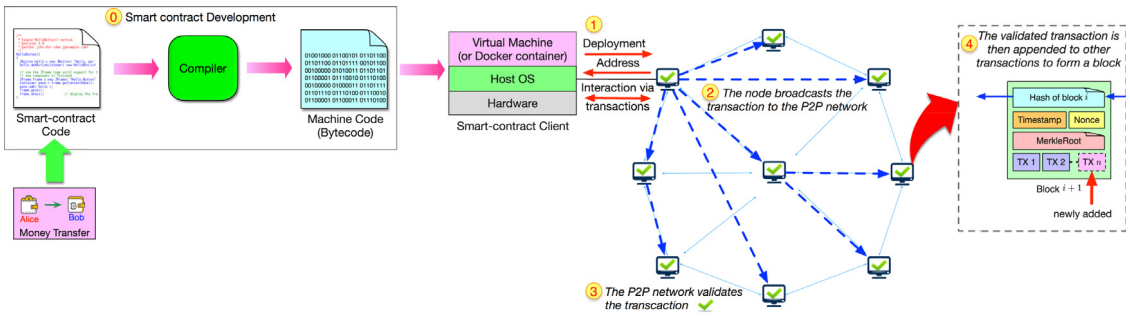

Fig. 6. Workflow of a smart contract.

```
// mortal.sol
pragma solidity ^0.4.0;
contract mortal {
    /* Define variable owner of the type address */
    address owner;
    /* This function is executed at initialization */
     constructor() internal { owner = msg.sender; }
    /* Function to send 500 wei to receiver's address; 10^18
         wei=1 ether; */
    function fundtransfer(address receiver) public { if
         (msg.sender == owner) {receiver.transfer(500);} }
}
```

Fig. 7. Example of smart contract written in Solidity in Ethereum.

public sector to sharing economy. In the future, Corda, EOS, Stellar and RSK and their derivatives may support more general applications.

4.3. Example of developing a smart contract

We next show how to develop and deploy a smart contract. Take a contract of money transfer between Alice and Bob as an example as shown in Fig. 6. After several rounds of negotiations, the agreement between Alice and Bob reaches. Then the agreement is implemented by a smart contract language (*e.g.*, Solidity in Ethereum and Golang in Fabric). The smart contract code is next compiled via a compiler (*e.g.*, solc for Solidity), which generates machine code (or bytecode) running on top of either virtual machines (*e.g.*, EVM, JVM) or Docker containers at a smart contract client. The smart contract client is essentially connected through a peer-to-peer network. After the smart contract is deployed across the blockchain network, a unique contract address is returned to the client to support the future interactions. Thereafter, users can interact with the blockchain network via executing transactions in the smart contract (*e.g.*, deducting the specified amount of money from Alice's digital wallet and increasing the corresponding amount of money in Bob's wallet). It is worth mentioning that each transaction needs to be validated across the blockchain network via the consensus algorithms as shown in Fig. 6. The validated transaction is then appended to the list of transactions. Since every node has a copy of the updated blockchain, it is difficulty to falsify the blockchain data.

Coding Sample. The syntax of Solidity is similar to JavaScript and it also supports inheritance and user-defined types. Fig. 7 shows an example of a smart contract written in Solidity.

5. Applications of smart contract

Smart contracts have a broad spectrum of applications ranging from Internet of Things to sharing economy. In particular, we roughly categorize major smart contract applications into six types as shown in Fig. 8. We next describe them in details.

5.1. Internet of things

Internet of things (IoT) that is one of the most promising technologies can support numerous applications including supply chain management, inventory control systems, retailers, access control, libraries, e-health systems, industrial Internet [97–99]. The main initiative of IoT is to integrate "*smart*" objects (*i.e.*, "things") into the Internet and to provide various services to users [100]. IoT has been proposed to automate various business transactions in an implicit way.

With the integration with smart contracts, the potentials of IoT can be unleashed. Take industrial manufacturing as an example. Most current manufacturers maintain their IoT ecosystems in a centralized manner. For instance, firmware updates can only be obtained at the central server *manually* by various IoT devices through querying from devices to the server. Smart contracts offer an automatic solution to this problem [101]. Manufacturers can place firmware update hashes on smart contracts deployed on blockchains distributed throughout the whole network. Devices can then obtain the firmware hashes from smart contracts automatically. In this way, resources are greatly saved.

Smart contracts can also bring benefits to IoT e-business model. For example, the traditional e-business model often requires a third party serving as an agent to complete the payment. However, this centralized payment is costly and cannot fully utilize advantages of IoT. In [4], Distributed autonomous Corporations (DACs) were proposed to automate transactions, in which there are no traditional roles like governments or companies involved with the payment. Being implemented by smart contracts, DACs can work automatically without human intervention. Moreover, smart contracts can also help to speed up conventional supply chains. For example, the marriage of supply chains with smart contracts can automate contractual rights and obligations during the payment and the delivery of goods while all the parties in the whole process are trustful.

5.2. Distributed system security

Smart contracts can bring benefits in improving the security of distributed systems. Distributed Denial-of-Service (DDoS) attacks are one of major security threats in computer networks. Attackers flood the targeted machine with superfluous requests to overload systems, consequently interrupting or suspending Internet services [102]. Recently, a collaborative mechanism was proposed to mitigate DDoS attacks [103]. Compared with traditional solutions, this scheme that is based on smart contracts can tackle the attacks in a fully decentralized manner. In particular, once a server is attacked, the IP addresses of attackers will be automatically stored in a smart contract. In this manner, other nodes will be informed of the addresses of attackers. Furthermore, other security polices will be immediately enforced, *e.g.*, filtering the traffic from the malicious users.

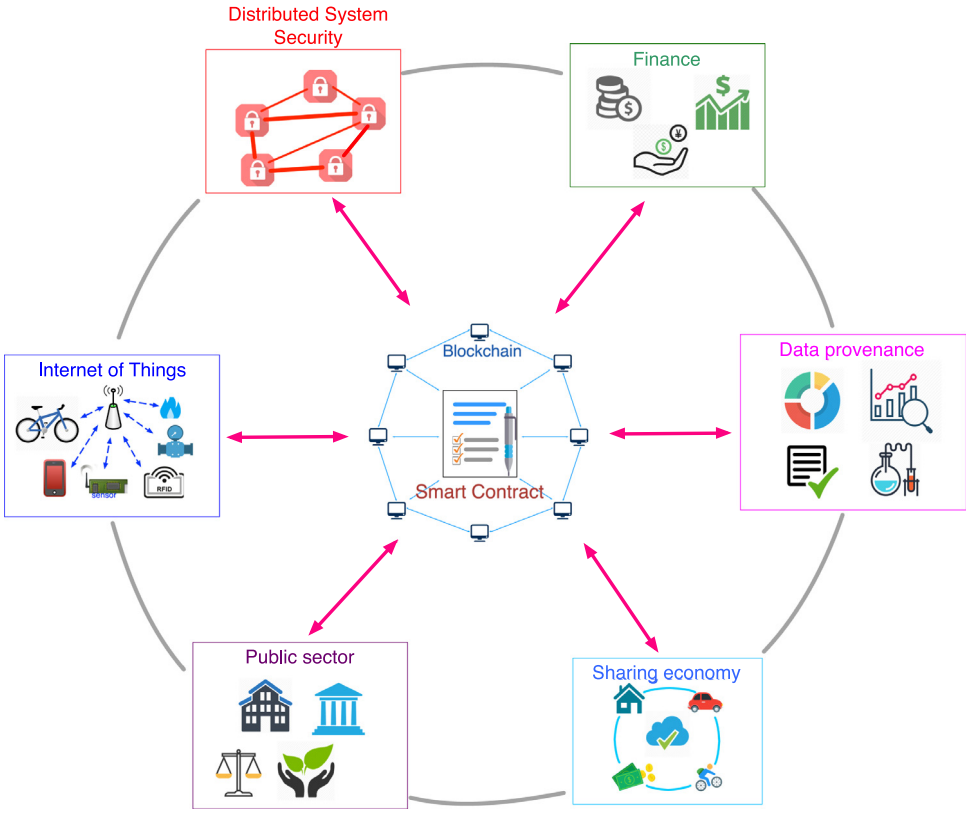

Fig. 8. Smart contract applications.

Cloud computing is a promising technology to offer a ubiquitous access of a shared pool of computing and storage resources to users [104]. Generally, users can purchase cloud services from trustful cloud service providers (CSPs). However, how to verify the trustfulness of CSPs becomes a challenge since CSPs often collude with each other to earn more profits. Dong et al. [105] proposed a solution based on game theory and smart contracts. The main idea of this approach is to let a client ask two cloud servers to compute the same task. During this process, smart contracts are used to stimulate tension, betrayal and distrust between the clouds. In this way, users can easily determine the rational clouds that will not collude and cheat. Experiments based on the contracts written in Solidity on the official Ethereum network were also conducted to verify the effectiveness of this proposal.

Moreover, brokers are typically used in cloud computing. Users' requests are checked by a broker to match with providers' services. However, both users and service providers must trust the broker. Once the broker is hijacked or compromised, both the parties become untrustful. Recently, Scoca et al. [9] proposed to use smart contracts to avoid the usage of brokers. The main idea of their approach is to use distributed Service-Level-Agreements for Clouds (dSLAC) [106] to specify the needs via smart contracts. Meanwhile, a utility function that evaluates the agreements according to both parties' preferences was proposed to solve the mismatching problem.

5.3. Finance

Smart contracts can potentially reduce financial risks, cut down administration and service costs and improve the efficiency of financial services. We next explain the benefits of smart contracts in the following typical financial services.

- *Capital markets and investment banking.* The traditional capital markets have suffered from the long settlement cycles. Smart contracts can significantly shorten the settlement period from 20 days or more to 6 to 10 days consequently increasing attractiveness to customers. As a result, it is predicted in [107] that it can bring 5% to 6% demand growth in the future and lead to additional income.
- *Commercial and retail banking.* In additional to capital markets, the adoption of smart contracts can also bring benefits to the mortgage loan industry [108]. Conventional mortgage loans are typically complicated in the origination, funding and servicing processes, consequently causing extra costs and delays. Smart contracts can potentially reduce the costs and the delays through automating the mortgage processes with the digitization of legal documents in blockchains.
- *Insurance.* The application of smart contracts in the insurance industry can also reduce the processing overheads and save the costs especially in claim handling [109]. Take the motor insurance as an example. There are multiple parties in a motor insurance: insurer, customers, garages, transport providers and hospitals [114]. Smart contracts can automate the settlement of claims by sharing legal documents in the distributed ledger consequently improving the efficiency, reducing the claim processing time and saving costs. For another example, the insurance giant AXA has launched its insurance for flight delay based on Ethereum smart contracts. Passengers who purchase flight insurances will automatically sign a smart contract, which connects to the global air traffic database. If the system notices a flight delay of over two hours, it will trigger a function in the smart contract, thereby passengers being paid immediately.

Table 5
Comparison of smart contract applications.

Application	Benefits	Use cases
Internet of Things [4,101]	✓Reducing the cost for maintaining central server ✓Automating P2P business trading ✓Reducing cost for trusted third parties	(1) IoT device firmware auto-updating (2) Supply chains speeding up
Distributed Systems Security [9,103,105]	✓Sharing attack list quickly and reliably ✓Verifying the trustfulness of cloud service providers ✓Avoiding usage of brokers	(1) Mitigating DDoS attack in computer networks (2) Cloud computing
Finance [107–109]	✓Reducing financial risks ✓Lowering administration and service costs ✓Improving efficiency of financial services	(1) Capital markets and investment banking (2) Commercial and retail banking (3) Insurance
Data Provenance [110,111]	✓Capturing malicious data falsification ✓Improving data reliability ✓Preserving privacy	(1) Scientific research (2) Public health (3) Cloud data provenance
Sharing Economy [3,112,113]	✓Reducing consumer costs ✓Reducing cost for trusted third parties ✓Preserving privacy	(1) Item sharing (2) P2P automatic payment systems (3) Currency exchange platforms
Public sector [5,7,8]	✓Preventing data fraudulence ✓Data transparency of public information ✓Preserving privacy	(1) E-voting systems (2) Personal reputation systems (3) Smart property exchange platforms

5.4. Data provenance

In addition to financial services, smart contracts can also be used to ensure information quality in scientific research and public health. It is reported in [115] that fabrication or falsification of data in clinical trials have occurred frequently in recent years. For another example, one paper published in Nature in 2009 was reported to contain fraud data conducted by Haruko Obokata [116]. The fabricated data can either mislead the ongoing research directions or hamper the recovery of patients. Consequently, it can seriously undermine the scientific and public trust.

Data provenance has been subsequently proposed to mitigate this problem. The main idea of data provenance is to store the meta-data information of data origin, derivation and transformation. However, there are a number of challenges in enforcing data provenance. For example, most provenance logging tools such as Progger [117] and Trusted Platform Module (TPM) [118] store data activities along with privacy-sensitive information (*e.g.*, user ID, accessing time and user roles). How to preserve the privacy information is a challenge. Ramachandran and Kantarcioglu [110] proposed a data provenance system based on smart contracts and blockchains. Researchers can submit their encrypted data to this system. When there are any data changes, smart contracts will be invoked to track the transformations made to the data. In this manner, any malicious falsifications of data can be captured.

In addition, Liang et al. [111] proposed ProvChain to collect and verify cloud data provenance. The main idea of ProvChain is to embed provenance data into blockchain transactions so that any data modifications are traceable. ProvChain consists of three procedures: provenance data collection, provenance data storage and provenance data validation. Experimental results demonstrate that ProvChain offers tamper-proof data provenance, privacy-preservation and data reliability.

Moreover, smart contract can be used to protect *intellectual property* of creative digital media. For example, Fig. 9 shows an example of this application. Each digital product is embedded with a unique digital watermark (such as buyer's digital wallet address and product ID). If there is any infringement (e.g., the buyer sells the digital product to others without the permission from the creator), the law enforcement officer can trace the illegal file with the original file via extracting the digital watermark and comparing the digital wallet address with buyer's. As a result, the infringement of property-right can be easily identified. The whole procedure can be achieved through smart contracts and blockchains.

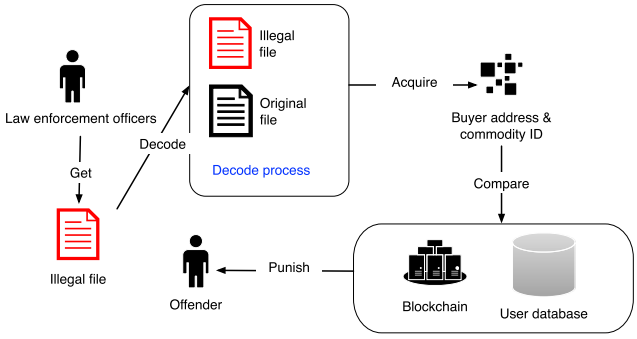

Fig. 9. Smart contract to protect intellectual property.

5.5. Sharing economy

The sharing economy brings many benefits such as reducing consumer costs by borrowing and recycling items, improving resource usage, enhancing quality of service, lowering the environment impacts [119]. However, most current sharing economy platforms are suffering from high transaction costs of customers, privacy exposure and unreliability of trusted third parties due to the centralization. Smart contracts can potentially reshape sharing economy by decentralizing sharing economy platforms.

Bogner et al. [3] proposed a novel sharing economy platform based on Ethereum smart contracts. In particular, this system allows users to register and share their items without a trusted third party. Meanwhile, personal information is also privacy-preserved. The practical implementation also verifies the effectiveness of the system. In addition, the fusion of Internet of Things (IoT) and smart contracts can also advance sharing economy applications. Huckle et al. [112] discussed the integration of IoT with blockchains to develop sharing economy applications such as peer-to-peer automatic payment systems, traveling systems, digital assets management and currency exchange platforms.

Meanwhile, a privacy respecting approach was proposed in [113] for blockchain-based sharing economy applications. This scheme mainly solves the privacy leakage problem of blockchain-based systems due to the public openness of blockchains. In particular, a zero-knowledge approach was applied to this system. Realistic implementation also demonstrates the effectiveness of the proposed mechanism.

5.6. Public sector

Smart contracts along with blockchain technology are also reshaping the public sector management. Blockchain can essentially prevent data fraudulence and provide the transparency of public information. Take a public bidding as an example. The integration of blockchains and smart contracts can prove identities of both bidders and bidding entities, automate the bidding process, provide auditing and reviewing supports.

There are several challenges in e-voting systems, such as user identity verification and user privacy preservation (or voting anonymity). Smart contracts also offer the solution to e-voting systems. A blockchain-based voting system named *Follow My Vote*[8] was proposed to verify user identities without the disclosure of user privacy. However, it still relies on a trusted third authority to shuffle the voters so as not to reveal user privacy. McCorry et al. [5] utilized the knowledge of self-tallying voting protocols (*i.e.*, voters can count the votes without a trusted third party) to build a fair voting system based on smart contracts. In this way, votes can be kept privately while user identities are verifiable at the same time.

Smart contracts can also be used to establish personal digital identity and reputation. For example, Tsinghua University User Reputation System (TURS) [8] is an online identity management system based on smart contracts. The TURS profile of a person is based on three aspects: personal reputation, online reputation and professional reputation. Users can protect their private information via smart contracts that grant access permissions to other users by programmable clauses (statements). Meanwhile, all the transactions that are recorded into blockchains cannot be tampered with or removed.

Hillbom and Tillström [7] proposed a smart property ownership exchange protocol based on the smart property concept firstly proposed by Szabo [120]. In this protocol, each party in the transaction communicates with each other via Bitmessage [121]. After the negotiation of the trading details (*e.g.*, a car's digital certificate issued by its manufacturer), the buyer constructs and signs a raw transaction that reassigns the property ownership to the buyer himself/herself. After the signed transaction is sent to the seller, the seller then checks the transaction information. If it is correct, the seller signs on the received transaction and broadcasts it publicly. Moreover, to ensure the consistency, the whole ownership transfer process has to be conducted in an atomic way. In other words, any failure during the whole process will abort the whole ownership transfer process. For example, if the seller does not sign the transaction, he/she will not get the funds from the buyer. Moreover, Li et al. [122] proposed a secure energy trading system based on consortium blockchain technology. In particular, a credit-based payment scheme was proposed to support fast energy trading without a trusted intermediary.

Summary. Table 5 compares the smart contract applications. As shown in Table 5, smart contracts can bring numerous benefits for the aforementioned applications. In summary, smart contracts have the merits like reducing the dependence on the trusted third parties, lowering the cost, improving the data reliability and offering privacy-preservation.

6. Conclusion

This article presents an overview on the state-of-the-art of smart contracts. In particular, we first provide a brief review on smart contract and blockchain technologies. We then point out the challenges in smart contracts in different aspects of creation, deployment, execution, completion of smart contracts. Meanwhile, we also discuss the recent advances in solving these challenges. We next compare several major smart contract platforms. Moreover, we categorize smart contract applications and enumerate several typical use cases in each category of applications. In summary, we hope this paper will serve as a guidance to developing secure and scalable smart contract applications and promote the evolvement of blockchain technologies.

On top of blockchains, smart contract is developing rapidly albeit there are still a number of challenges to be addressed. Most of current research topics on smart contract focus on programming language, security and privacy issues while the proliferation of blockchain and smart contract applications also poses new challenges. Like other computer software tools, smart contracts also contain a number of bugs which are generated unintentionally or mischievously. However, detecting and identifying these bugs will require extensive efforts in aspects of software engineering and data analytics. In addition, although enterprise practitioners lack of knowledge in computing programming, they have the expertise in operational technology and law making (i.e., making contracts), which however is the deficiency of computer programmers. How to fill the gap between operational technology (OT) and information technology (IT) is of great importance to the development of smart contracts. The integration of software technology, natural language processing and artificial intelligence is a possible remedy to this challenge in the future.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The work described in this paper was supported by the National Key Research and Development Program, China (2016-YFB1000101), the National Natural Science Foundation of China under (61472338) and the Fundamental Research Funds for the Central Universities, China. Imran's work is supported by the Deanship of Scientific Research at King Saud University, Saudi Arabia through the research group project number RG-1435-051. The authors would like to thank anonymous reviewers who have provided constructive comments greatly improving the paper.

References

[1] N. Szabo, The idea of smart contracts, in: Nick Szabo's Papers and Concise Tutorials, 1997, URL http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart_contracts_2.html.

[2] H.-N. Dai, Z. Zheng, Y. Zhang, Blockchain for internet of things: A survey, IEEE Internet Things J. 6 (5) (2019) 8076–8094, http://dx.doi.org/10.1109/JIOT.2019.2920987.

[3] A. Bogner, M. Chanson, A. Meeuw, A decentralised sharing app running a smart contract on the ethereum blockchain, in: Proceedings of the 6th International Conference on the Internet of Things, 2016, pp. 177–178.

[4] Y. Zhang, J. Wen, An IoT electric business model based on the protocol of bitcoin, in: Proceedings of 18th International Conference on Intelligence in Next Generation Networks, ICIN, 2015, pp. 184–191.

[5] P. McCorry, S.F. Shahandashti, F. Hao, A smart contract for boardroom voting with maximum voter privacy, IACR Cryptol. ePrint Archive 2017 (2017) 110.

[6] L. Luu, Y. Velner, J. Teutsch, P. Saxena, SMART POOL: Practical decentralized pooled mining, in: 26th USENIX Security Symposium, USENIX Security, 2017, pp. 1409–1426.

[7] E. Hillbom, T. Tillström, Applications of Smart Contracts and Smart Property Utilizing Blockchains (M.Sc. thesis in computer science), Chalmers University of Technology and University of Gothenburg, Sweden, 2016.

[8] Follow My Vote https://followmyvote.com/.

[8] A. Yasin, L. Liu, An online identity and smart contract management system, in: Proceedings of 40th Annual Computer Software and Applications Conference, COMPSAC, vol. 2, 2016, pp. 192–198.

[9] V. Scoca, R.B. Uriarte, R. De Nicola, Smart contract negotiation in cloud computing, in: Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on, IEEE, 2017, pp. 592–599.

[10] J. Wan, J. Li, M. Imran, D. Li, et al., A blockchain-based solution for enhancing security and privacy in smart factory, IEEE Trans. Ind. Inf. (2019).

[11] S. Moin, A. Karim, Z. Safdar, K. Safdar, E. Ahmed, M. Imran, Securing iots in distributed blockchain: Analysis, requirements and open issues, Future Gener. Comput. Syst. (2019).

[12] Z. Zheng, S. Xie, H. Dai, X. Chen, H. Wang, An overview of blockchain technology: Architecture, consensus, and future trends, in: 2017 IEEE International Congress on Big Data, BigData Congress, IEEE, 2017, pp. 557–564.

[13] S. Omohundro, Cryptocurrencies, smart contracts, and artificial intelligence, AI Matters 1 (2) (2014) 19–21.

[14] X. Li, P. Jiang, T. Chen, X. Luo, Q. Wen, A survey on the security of blockchain systems, Future Gener. Comput. Syst. (2017).

[15] N. Atzei, M. Bartoletti, T. Cimoli, A survey of attacks on ethereum smart contracts (SoK), in: Proceddings of International Conference on Principles of Security and Trust, 2017, pp. 164–186.

[16] K. Delmolino, M. Arnett, A. Kosba, A. Miller, E. Shi, Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab, in: Proceedings of International Conference on Financial Cryptography and Data Security, 2016, pp. 79–94.

[17] D. Harz, W. Knottenbelt, Towards safer smart contracts: A survey of languages and verification methods, 2018, arXiv preprint arXiv:1809.09805.

[18] M. Bartoletti, L. Pompianu, An empirical analysis of smart contracts: platforms, applications, and design patterns, in: International Conference on Financial Cryptography and Data Security, Springer, 2017, pp. 494–509.

[19] M. Alharby, A. van Moorsel, Blockchain-based smart contracts: A systematic mapping study, 2017, arXiv preprint arXiv:1710.06372.

[20] D. Macrinici, C. Cartofeanu, S. Gao, Smart contract applications within blockchain technology: A systematic mapping study, Telemat. Inform. (2018).

[21] S. Wang, Y. Yuan, X. Wang, J. Li, R. Qin, F.-Y. Wang, An overview of smart contract: architecture, applications, and future trends, in: 2018 IEEE Intelligent Vehicles Symposium, IV, IEEE, 2018, pp. 108–113.

[22] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, Y. Zhou, Detecting ponzi schemes on ethereum: Towards healthier blockchain technology, in: Proceedings of the 2018 World Wide Web Conference on World Wide Web, International World Wide Web Conferences Steering Committee, 2018, pp. 1409–1418.

[23] S. Nakamoto, Bitcoin: A peer-to-peer electronic cash system, 2008.

[24] M. Castro, B. Liskov, et al., Practical byzantine fault tolerance, in: OSDI, vol. 99, 1999, pp. 173–186.

[25] S. King, S. Nadal, Ppcoin: Peer-to-peer crypto-currency with proof-of-stake, 2012, Self-Published Paper, August 19.

[26] Z. Zheng, S. Xie, H.-N. Dai, X. Chen, H. Wang, Blockchain challenges and opportunities: A survey, Int. J. Web Grid Serv. 14 (4) (2018) 352–375.

[27] D. Tapscott, A. Tapscott, Blockchain Revolution: How the Technology Behind Bitcoin Is Changing Money, Business, and the World, first ed., Penguin, 2016.

[28] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, R.H. Deng, CrowdBC: A blockchain-based decentralized framework for crowdsourcing, IEEE Trans. Parallel Distrib. Syst. (2018) http://dx.doi.org/10.1109/TPDS.2018.2881735.

[29] J. Ream, Y. Chu, D. Schatsky, Upgrading Blockchains: Smart Contract Use Cases in Industry, Deloitte Press, 2016, URL https://www2.deloitte.com/insights/us/en/focus/signals-for-strategists/using-blockchain-for-smart-contracts.html.

[30] F. Idelberger, G. Governatori, R. Riveret, G. Sartor, Evaluation of logic-based smart contracts for blockchain systems, in: International Symposium on Rules and Rule Markup Languages for the Semantic Web, RuleML, Springer, 2016, pp. 167–183.

[31] C. Sillaber, B. Waltl, Life cycle of smart contracts in blockchain ecosystems, Datenschutz Datensicherheit - DuD 41 (8) (2017) 497–500.

[32] R. Koulu, Blockchains and online dispute resolution: smart contracts as an alternative to enforcement, SCRIPTed 13 (2016) 40.

[33] Y. Zhou, D. Kumar, S. Bakshi, J. Mason, A. Miller, M. Bailey, Erays: reverse engineering ethereum's opaque smart contracts, in: 27th {USENIX} Security Symposium {USENIX} Security 18, 2018, pp. 1371–1385.

[34] C.K. Frantz, M. Nowostawski, From institutions to code: towards automated generation of smart contracts, in: Proceedings of IEEE International Workshops on Foundations and Applications of Self Systems, 2016, pp. 210–215.

[35] G. Ciatto, R. Calegari, S. Mariani, E. Denti, A. Omicini, From the Blockchain to Logic Programming and Back: Research Perspectives, WOA, 2018.

[36] T. Kasampalis, D. Guth, B. Moore, T. Serbanuta, V. Serbanuta, D. Filaretti, G. Rosu, R. Johnson, IELE: An Intermediate-Level Blockchain Language Designed and Implemented Using Formal Semantics, Tech. Rep., 2018.

[37] C. Lattner, V. Adve, LLVM: A compilation framework for lifelong program analysis & transformation, in: Proceedings of the International Symposium on Code Generation and Optimization: Feedback-Directed and Runtime Optimization, IEEE Computer Society, 2004, p. 75.

[38] M. Coblenz, Obsidian: A safer blockchain programming language, in: Proceedings of the 39th International Conference on Software Engineering Companion, ICSE-C '17, 2017, pp. 97–99.

[39] A. Mavridou, A. Laszka, Tool demonstration: Fsolidm for designing secure ethereum smart contracts, in: International Conference on Principles of Security and Trust, Springer, 2018, pp. 270–277.

[40] C. Liu, H. Liu, Z. Cao, Z. Chen, B. Chen, B. Roscoe, ReGuard: finding reentrancy bugs in smart contracts, in: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ACM, 2018, pp. 65–68.

[41] J. Bonneau, J. Clark, S. Goldfeder, On bitcoin as a public randomness source, IACR Cryptol. ePrint Archive 2015 (2015) 1015.

[42] A.K. Lenstra, B. Wesolowski, A random zoo: sloth, unicorn, and trx, IACR Cryptol. ePrint Archive 2015 (2015) 366.

[43] B. Bünz, S. Goldfeder, J. Bonneau, Proofs-of-delay and randomness beacons in ethereum, in: IEEE Security and Privacy on the Blockchain, IEEE S&B, 2017.

[44] T. Chen, X. Li, X. Luo, X. Zhang, Under-optimized smart contracts devour your money, in: Proceedings of 24th International Conference on Software Analysis, Evolution and Reengineering, SANER, 2017, pp. 442–446.

[45] T. Chen, Z. Li, H. Zhou, J. Chen, X. Luo, X. Li, X. Zhang, Towards saving money in using smart contracts, in: 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results, ICSE-NIER, IEEE, 2018, pp. 81–84.

[46] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, A. Hobor, Making smart contracts smarter, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 254–269.

[47] E. Albert, P. Gordillo, B. Livshits, A. Rubio, I. Sergey, EthIR: A framework for high-level analysis of ethereum bytecode, in: International Symposium on Automated Technology for Verification and Analysis, Springer, 2018, pp. 513–520.

[48] M. Knecht, B. Stiller, SmartDEMAP: A smart contract deployment and management platform, in: Proceedings of International Conference on Autonomous Infrastructure, Management and Security, 2017, pp. 159–164.

[49] N. Grech, M. Kong, A. Jurisevic, L. Brent, B. Scholz, Y. Smaragdakis, Madmax: Surviving out-of-gas conditions in ethereum smart contracts, in: Proceedings of the ACM on Programming Languages, vol. 2, no. OOPSLA, ACM, 2018, p. 116.

[50] J. Krupp, C. Rossow, teether: Gnawing at ethereum to automatically exploit smart contracts, in: 27th {USENIX} Security Symposium, {USENIX} Security 18, 2018, pp. 1317–1333.

[51] L. Brent, A. Jurisevic, M. Kong, E. Liu, F. Gauthier, V. Gramoli, R. Holz, B. Scholz, Vandal: A scalable security analysis framework for smart contracts, 2018, arXiv preprint arXiv:1809.03981.

[52] N. Grech, L. Brent, B. Scholz, Y. Smaragdakis, Gigahorse: Thorough, declarative decompilation of smart contracts.

[53] S. Amani, M. Bégel, M. Bortin, M. Staples, Towards verifying ethereum smart contract bytecode in Isabelle/HOL, in: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs, ACM, 2018, pp. 66–77.

[54] K. Bhargavan, A. Delignat-Lavaud, C. Fournet, A. Gollamudi, G. Gonthier, N. Kobeissi, N. Kulatova, A. Rastogi, T. Sibut-Pinote, N. Swamy, S. Zanella-Béguelin, Formal verification of smart contracts: short paper, in: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS, 2016.

[55] N. Swamy, C. Hriţcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J.-K. Zinzindohoue, S. Zanella-Béguelin, Dependent types and multi-monadic effects in F*, in: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL, 2016, pp. 256–270.

[56] S. Kalra, S. Goel, M. Dhawan, S. Sharma, Zeus: Analyzing safety of smart contracts, in: 25th Annual Network and Distributed System Security Symposium, NDSS, 2018, pp. 18–21.
[57] K.L. McMillan, Interpolants and symbolic model checking, in: International Workshop on Verification, Model Checking, and Abstract Interpretation, Springer, 2007, pp. 89–90.
[58] H. Liu, C. Liu, W. Zhao, Y. Jiang, J. Sun, S-gram: towards semantic-aware security auditing for ethereum smart contracts, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, ACM, 2018, pp. 814–819.
[59] T.H.-D. Huang, Hunting the ethereum smart contract: Color-inspired inspection of potential attacks, 2018, arXiv preprint arXiv:1807.01868.
[60] A. Tann, X.J. Han, S.S. Gupta, Y.-S. Ong, Towards safer smart contracts: A sequence learning approach to detecting vulnerabilities, 2018, arXiv preprint arXiv:1811.06632.
[61] J. Charlier, S. Lagraa, R. State, J. François, Profiling smart contracts interactions tensor decomposition and graph mining, in: Proceedings of the Second Workshop on MIning DAta for Financial Applications, MIDAS 2017, 2017, pp. 31–42.
[62] M. Fröwis, R. Böhme, In code we trust? in: Data Privacy Management, Cryptocurrencies and Blockchain Technology, Springer, 2017, pp. 357–372.
[63] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, A. Hobor, Finding the greedy, prodigal, and suicidal contracts at scale, in: Proceedings of the 34th Annual Computer Security Applications Conference, ACM, 2018, pp. 653–663.
[64] Y. Fu, M. Ren, F. Ma, Y. Jiang, H. Shi, J. Sun, EVMFuzz: Differential fuzz testing of ethereum virtual machine, 2019, arXiv preprint arXiv:1903.08483.
[65] F. Zhang, E. Cecchetti, K. Croman, A. Juels, E. Shi, Town crier: An authenticated data feed for smart contracts, in: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, 2016, pp. 270–282.
[66] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, A. Kastania, Astraea: A decentralized blockchain oracle, 2018, arXiv preprint arXiv:1808.00528.
[67] S. Ellis, A decentralized oracle network steve ellis, ari juels, and sergey nazarov, 2017.
[68] A. Mavridou, A. Laszka, Designing secure ethereum smart contracts: A finite state machine based approach, 2017, arXiv preprint arXiv:1711.09327.
[69] C. Natoli, V. Gramoli, The blockchain anomaly, in: 2016 IEEE 15th International Symposium on Network Computing and Applications, NCA, IEEE, 2016, pp. 310–317.
[70] T. Dickerson, P. Gazzillo, M. Herlihy, E. Koskinen, Adding concurrency to smart contracts, in: Proceedings of the ACM Symposium on Principles of Distributed Computing, PODC, ACM, 2017, pp. 303–312.
[71] I. Sergey, A. Hobor, A concurrent perspective on smart contracts, in: International Conference on Financial Cryptography and Data Security, vol. 2017, 2017.
[72] P.S. Anjana, S. Kumari, S. Peri, S. Rathor, A. Somani, An efficient framework for concurrent execution of smart contracts, 2018, arXiv preprint arXiv:1809.01326.
[73] S. Bragagnolo, H. Rocha, M. Denker, S. Ducasse, Smartinspect: solidity smart contract inspector, in: 2018 International Workshop on Blockchain Oriented Software Engineering, IWBOSE, IEEE, 2018, pp. 9–18.
[74] A. Kosba, A. Miller, E. Shi, Z. Wen, C. Papamanthou, Hawk: The blockchain model of cryptography and privacy-preserving smart contracts, in: Proceedings of 2016 IEEE Symposium on Security and Privacy, SP, 2016, pp. 839–858.
[75] G. Zyskind, O. Nathan, A. Pentland, Enigma: Decentralized computation platform with guaranteed privacy, 2015, arXiv preprint arXiv:1506.03471.
[76] M. Apostolaki, G. Marti, J. Müller, L. Vanbever, SABRE: Protecting bitcoin against routing attacks, 2018, arXiv preprint arXiv:1808.06254.
[77] M. Bartoletti, S. Carta, T. Cimoli, R. Saia, Dissecting ponzi schemes on ethereum: identification, analysis, and impact, 2017, arXiv preprint arXiv:1703.03779.
[78] M. Bartoletti, B. Pes, S. Serusi, Data mining for detecting bitcoin ponzi schemes, in: 2018 Crypto Valley Conference on Blockchain Technology, CVCBT, IEEE, 2018, pp. 75–84.
[79] C.F. Torres, M. Steichen, The art of the scam: Demystifying honeypots in ethereum smart contracts, 2019, arXiv preprint arXiv:1902.06976.
[80] E. Ostrom, Collective action and the evolution of social norms, J. Nat. Resour. Policy Res. 6 (4) (2014) 235–252.
[81] X. Li, P. Jiang, T. Chen, X. Luo, Q. Wen, A survey on the security of blockchain systems, Future Gener. Comput. Syst. (2017) http://dx.doi.org/10.1016/j.future.2017.08.020.
[82] J. Bonneau, J. Clark, S. Goldfeder, On bitcoin as a public randomness source, IACR Cryptol. ePrint Archive 2015 (2015) 1015.
[83] The dao, the hack, the soft fork and the hard fork, 2017, URL https://www.cryptocompare.com/coins/guides/the-dao-the-hack-the-soft-fork-and-the-hard-fork/.
[84] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Buenzli, M. Vechev, Securify: Practical security analysis of smart contracts, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, ACM, 2018, pp. 67–82.
[85] A. Silberschatz, H.F. Korth, S. Sudarshan, Database System Concepts, sixth ed., McGraw-Hill Education, 2010.
[86] G. Bracha, D. Ungar, Mirrors: design principles for meta-level facilities of object-oriented programming languages, in: ACM SIGPLAN Notices, vol. 39, ACM, 2004, pp. 331–344.
[87] D. Ron, A. Shamir, Quantitative analysis of the full bitcoin transaction graph, in: Financial Cryptography and Data Security: 17th International Conference, FC 2013, Okinawa, Japan, Springer Berlin Heidelberg, 2013, pp. 6–24.
[88] M. Apostolaki, A. Zohar, L. Vanbever, Hijacking bitcoin: Routing attacks on cryptocurrencies, in: Security and Privacy (SP), IEEE Symposium on, IEEE, 2017, pp. 375–392.
[89] V. Buterin, et al., Ethereum White Paper, Ethereum, 2013, URL https://www.ethereum.org/.
[90] C. Cachin, Architecture of the hyperledger blockchain fabric, in: Workshop on Distributed Cryptocurrencies and Consensus Ledgers, 2016.
[91] R.G. Brown, The corda platform: An introduction, 2018, URL https://www.corda.net/content/corda-platform-whitepaper.pdf.
[92] D. Mazieres, The Stellar Consensus Protocol: A Federated Model for Internet-Level Consensus, Stellar Development Foundation, 2016, URL https://www.stellar.org/papers/stellar-consensus-protocol.pdf.
[93] S.D. Lerner, Rootstock whitepaper, 2015, URL https://docs.rsk.co/RSK_White_Paper-Overview.pdf.
[94] T. Bocek, B. Stiller, Smart contracts – blockchains in the wings, in: Digital Marketplaces Unleashed, Springer, Berlin, Heidelberg, 2018, pp. 169–184.
[95] H. Howard, M. Schwarzkopf, A. Madhavapeddy, J. Crowcroft, Raft refloated: Do we have consensus? SIGOPS Oper. Syst. Rev. 49 (1) (2015) 12–21.
[96] E. IO, EOS. IO Technical White Paper, EOS. IO, 2017, (Accessed 18 December 2017). https://github.com/EOSIO/Documentation.
[97] R.A.A. Habeeb, F. Nasaruddin, A. Gani, I.A.T. Hashem, E. Ahmed, M. Imran, Real-time big data processing for anomaly detection: A survey, Int. J. Inf. Manage. (2018).
[98] H.A. Khattak, M.A. Shah, S. Khan, I. Ali, M. Imran, Perception layer security in internet of things, Future Gener. Comput. Syst. (2019).
[99] I. Yaqoob, E. Ahmed, M.H. ur Rehman, A.I.A. Ahmed, M.A. Al-garadi, M. Imran, M. Guizani, The rise of ransomware and emerging security challenges in the internet of things, Comput. Netw. 129 (2017) 444–458.
[100] D. Miorandi, S. Sicari, F.D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges, Ad Hoc Netw. 10 (7) (2012) 1497–1516.
[101] K. Christidis, M. Devetsikiotis, Blockchains and smart contracts for the internet of things, IEEE Access 4 (2016) 2292–2303.
[102] S. Mansfield-Devine, The growth and evolution of DDoS, Netw. Secur. 2015 (10) (2015) 13–20.
[103] B. Rodrigues, T. Bocek, A. Lareida, D. Hausheer, S. Rafati, B. Stiller, A blockchain-based architecture for collaborative ddos mitigation with smart contracts, in: Proceedings of International Conference on Autonomous Infrastructure, Management and Security, 2017, pp. 16–29.
[104] H. Wang, P. Shi, Y. Zhang, Jointcloud: A cross-cloud cooperation architecture for integrated internet service customization, in: 2017 IEEE 37th International Conference on Distributed Computing Systems, ICDCS, 2017, pp. 1846–1855, http://dx.doi.org/10.1109/ICDCS.2017.237.
[105] C. Dong, Y. Wang, A. Aldweesh, P. McCorry, A. van Moorsel, Betrayal, distrust, and rationality: Smart counter-collusion contracts for verifiable cloud computing, in: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS, ACM, 2017, pp. 211–227.
[106] R.B. Uriarte, F. Tiezzi, R. De Nicola, Dynamic SLAs for clouds, in: Proceedings of European Conference on Service-Oriented and Cloud Computing, 2016, pp. 34–49.
[107] B. Cant, A. Khadikar, A. Ruiter, J.B. Bronebakk, J. Coumaros, J. Buvat, A. Gupta, Smart contracts in financial services: Getting from hype to reality, Capgemini Consult. (2016) 1–26, URL https://www.capgemini.com/consulting-de/wp-content/uploads/sites/32/2017/08/smart_contracts_paper_long_0.pdf.

[108] Y. Guo, C. Liang, Blockchain application and outlook in the banking industry, Financ. Innov. 2 (1) (2016) 24.
[109] A. Tapscott, D. Tapscott, How blockchain is changing finance, Harv. Bus. Rev. 1 (2017).
[110] A. Ramachandran, D. Kantarcioglu, et al., Using blockchain and smart contracts for secure data provenance management, 2017, arXiv preprint arXiv:1709.10000.
[111] X. Liang, S. Shetty, D. Tosh, C. Kamhoua, K. Kwiat, L. Njilla, ProvChain: A blockchain-based data provenance architecture in cloud environment with enhanced privacy and availability, in: 2017 17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGRID, 2017, pp. 468–477.
[112] S. Huckle, R. Bhattacharya, M. White, N. Beloff, Internet of things, blockchain and shared economy applications, Procedia Comput. Sci. 98 (2016) 461–466.
[113] L. Xu, N. Shah, L. Chen, N. Diallo, Z. Gao, Y. Lu, W. Shi, Enabling the sharing economy: Privacy respecting contract based on public blockchain, in: Proceedings of the ACM Workshop on Blockchain, Cryptocurrencies and Contracts, ACM, 2017, pp. 15–21.
[114] W. Du, M.J. Atallah, Secure multi-party computation problems and their applications: a review and open problems, in: Proceedings of the 2001 Workshop on New Security Paradigms, 2001, pp. 13–22.
[115] S.L. George, Research misconduct and data fraud in clinical trials: prevalence and causal factors, Int. J. Clin. Oncol. 21 (1) (2016) 15–21.
[116] J. Rasko, C. Power, What pushes scientists to lie? The disturbing but familiar story of Haruko Obokata, The Guardian 18 (2015).
[117] R.K. Ko, M.A. Will, Progger: An efficient, tamper-evident kernel-space logger for cloud data provenance tracking, in: Cloud Computing (CLOUD), 2014 IEEE 7th International Conference on, IEEE, 2014, pp. 881–889.
[118] M.M.B. Taha, S. Chaisiri, R.K.L. Ko, Trusted tamper-evident data provenance, in: 2015 IEEE Trustcom/BigDataSE/ISPA, 2015, pp. 646–653.
[119] A. Taeihagh, Crowdsourcing, sharing economies and development, J. Dev. Soc. 33 (2) (2017) 191–222.
[120] N. Szabo, Formalizing and securing relationships on public networks, First Monday 2 (9) (1997).
[121] J. Warren, Bitmessage: A peer-to-peer message authentication and delivery system, in: White Paper, 2012, https://bitmessage.org/bitmessage.pdf.
[122] Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng, Y. Zhang, Consortium blockchain for secure energy trading in industrial internet of things, IEEE Trans. Ind. Inf. PP (99) (2017) 1, http://dx.doi.org/10.1109/TII.2017.2786307.

Zibin Zheng is a professor at Sun Yat-sen University, Guangzhou, China. He received Ph.D. degree from The Chinese University of Hong Kong in 2011. He received ACM SIGSOFT Distinguished Paper Award at ICSE'10, Best Student Paper Award at ICWS'10, and IBM Ph.D. Fellowship Award. His research interests include services computing, software engineering, and blockchain.

Shaoan Xie is a graduate student at Sun Yat-Sen University, China. He received his bachelor degree in Computer Science at Sun yat-sen University in 2016. His current research interests include blockchain and data mining.

Hong-Ning Dai is an Associate Professor in Faculty of Information Technology at Macau University of Science and Technology. He obtained the Ph.D. degree in Computer Science and Engineering from Department of Computer Science and Engineering at the Chinese University of Hong Kong. He has published more than 80 peer-reviewed papers in refereed journals and conferences. He is also a holder of 1 U.S. patent and 1 Australia innovation patent. He is the winner of Bank of China (BOC) Excellent Research Award of Macau University of Science and Technology in 2015. He also holds visiting positions at Department of Computer Science and Engineering, The Hong Kong University of Science and Technology and School of Electrical Engineering and Telecommunications, the University of New South Wales, respectively. He has served as a guest editor for IEEE Transactions on Industrial Informatics and an editor for International Journal of Wireless and Mobile Communication for Industrial Systems. His research interests include wireless networks, mobile computing, and distributed systems.

Weili Chen is currently working toward the PhD degree in the Department of Data and Computer Science, Sun Yat-Sen University, China. His research interests include blockchain and data mining.

Xiangping Chen is a research associate at Sun Yat-sen University, Guangzhou, China. She received the PhD degree in computer science and technology from Peking University, China, in 2010. Her research interests include data driven software engineering, blockchain, program comprehension.

Jian Weng is a professor and Executive Dean with College of Information Science and Technology in Jinan University. He received B.S. degree and M.S. degree at South China University of Technology in 2001 and 2004 respectively, and Ph.D. degree at Shanghai Jiao Tong University in 2008. His research areas include cryptography, system security, etc. He has published 80 papers in international conferences and journals such as CRYPTO, EUROCRYPT, ASIACRYPT, TCC, PKC, CT-RSA, IEEE TDSC, etc. He received the Young Scientists Fund of the National Natural Science Foundation of China in 2018. He received the first Cryptography Innovation Award from Chinese Association for Cryptologic Research (CACR), the Best Paper Award of 28 th Symposium on Cryptography and Information Security (SCIS 2011), and the National Excellent Teacher in Network Security. He served as General Co-Chair for SecureComm 2016, TPC Co-Chairs for RFIDsec'13 Asia and ISPEC 2011, and program committee members for more than 30 international cryptography and information security conferences. He also serves as associate editor of IEEE Transactions on Vehicular Technology.

Muhammad Imran is working as Assistant Professor in the College of Computer and Information Sciences, King Saud University (KSU) since 2011. He is also a Visiting Scientist at Iowa State University, USA. His research interest includes Internet of Things, Big Data Analytics, Intelligent Transportation Systems, Cloud and Edge computing, and Security and privacy. His research is financially supported by several grants and he has completed a number of international collaborative research projects with reputable universities. He has published more than one hundred research articles in top conferences and journals. European Alliance for Innovation (EAI) has appointed him as an Editor in Chief for EAI Transactions on Pervasive Health and Technology. He also serves as an associate editor for reputable international journals such as IEEE Communications Magazine, Future Generation Computer Systems, IEEE Access, Wireless Communication and Mobile Computing Journal (SCIE, Wiley), Ad Hoc & Sensor Wireless Networks Journal (SCIE), IET Wireless Sensor Systems, International Journal of Autonomous and Adaptive Communication Systems (Inderscience). He served/serving as a guest editor for more than a dozen special issues in journals such as IEEE Communications Magazine, Computer Networks (Elsevier), Future Generation Computer Systems (Elsevier), MDPI Sensors, International Journal of Distributed Sensor Networks (Hindawi), Journal of Internet Technology, and International Journal of Autonomous and Adaptive Communications Systems. He has been involved in more than seventy conferences and workshops in various capacities such as a chair, co-chair and technical program committee member.