



Blockchain-Based Software Architecture Development for Service Requirements With Smart Contracts

Yan Zhu, Qian Guo, and Hongjian Yin, University of Science and Technology Beijing

Kaitai Liang, Delft University of Technology

Stephen S. Yau, Arizona State University

We review the background, classification, specifications, and related aspects of using smart contracts in blockchain services and introduce a framework for generating a general software architecture in the context of contract-oriented programming.

Smart contracts, which enable programmers to perform flexible development across decentralized applications, have been considered the technical core of blockchain.¹ Most blockchain application developers have smart contract platforms in their distributed ledger services, such as Ethereum Virtual Machine, Bitcoin-based Rootstock, and Hyperledger Fabric. To date, smart contract applications have been

extensively used in areas ranging from cryptocurrency to finance, real estate, gaming/gambling, and health care. In particular, they have become an innovation engine in financial services, which make up one of the economy's most important and influential sectors.²

However, there has been no systematic study of the impact smart contracts have on developing a software architecture for financial services. In this article, we briefly review the background, classification, specifications, and related aspects of using smart contracts in such a context. Then, we introduce a framework for generating a general

software architecture using smart contracts in the framework of contract-oriented programming. In addition, we discuss a smart legal contract language and its compilation, deployment mechanism, and execution process.

STATE OF THE ART

The concept of smart contracts can be traced to “Smart Contracts: Building Blocks for Digital Markets,” written by Nick Szabo in 1994.³ His article defined a smart contract as a computerized transaction protocol that executes the terms of an agreement, creatively stating that “the basic idea of smart contracts is that many kinds of contractual terms (such as liens, bonding, delineation of property rights, etc.) can be embedded in the hardware and software we deal with, in such a way as to make breach of contract expensive (if desired, sometimes prohibitively so) for the breacher.” Szabo has not stopped exploring smart contracts. For example, his subsequent articles have suggested executings contracts for synthetic assets (such as derivatives and bonds).

Due to the lack of a credible execution environment, Szabo’s smart contract has not been applied to practical applications. Since Bitcoin was developed in 2008, researchers have realized that the currency’s underlying platform (blockchain) might provide a credible ecosystem for smart contracts.⁴ In a white paper,⁵ Ethereum explored the possibility of combining blockchain and smart contracts, refined the concept of smart contracts, and established a set of specifications and a framework. After being “accepted” by Ethereum, smart contracts formed a technical pillar of blockchain, and developers began to extend their use to various domains. Ethereum and Hyperledger are the most widely used blockchain platforms that

support smart contract development and deployment, and their execution mechanisms are well designed.

Smart contracts, from various perspectives, are understood and categorized differently and subject to alternating expectations. Traditionally, a contract is a document signed by specific parties, and it is an agreed framework for assigning rights and obligations that does not require mutual trust. By contrast, a smart contract refers to any computer protocol that conforms to an agreement between many parties.⁶ A smart contract can be processed by computers. Unlike algorithms executed on stand-alone machines, it requires two or more participants to cooperatively perform a task. Its execution should satisfy the participants’ agreement, demonstrating credibility and compliance as well as providing the technical means to ensure protocol verification, storage, dispute resolution, and so on. As opposed to a traditional paper contract, a smart contract’s intelligence is indirectly captured by the computerization of a multiparty protocol and the adoption of a corresponding guarantee technique.

From a legal perspective, a smart contract might be considered a computer procedure containing a set of predefined rules that may include the facilitation, verification, and execution of a binding agreement. This statement shows the legal characteristic of a smart contract.⁷ The protocol is an approach through which digital code is the manifestation of an intent declaration, with the purpose to promote, verify, and strengthen contract negotiation. Smart contracts are now widely defined as lines of computer code stored on a blockchain and automatically executed when predetermined terms and conditions are met; they are also known as *blockchain smart contracts*.⁸ From this perspective, it

can be concluded that smart contracts stored on a blockchain are automatically executed computer code that specifies the terms of an agreement between parties and is directly embedded in the blockchain. The execution of smart contracts does not require human intervention; therefore, it is called *self-executing* and *self-enforcing*.

It should be noted that smart contracts, as code representations of binding terms, are a set of computer functions and could be part of application software.⁹ Moreover, blockchain smart contracts are executed on computer networks, a process that does not require the participation of trusted parties among which correct execution is guaranteed by consensus. Thus, smart contracts can be understood as automatable and enforceable agreements that need no prior mediation and provide verification and self-execution.

SMART CONTRACT ENGINEERING TO DEVELOP SERVICE APPLICATIONS

From the perspective of software engineering, smart contracts are not only executable code on a blockchain but a complete software architecture that provides technical guidance for contract-oriented design, development, deployment, and execution, a process referred to as *smart contract engineering*. With various application requirements, smart contract engineering provides an effective way to develop and deploy contract-oriented software for services. Differing from traditional software engineering models, for example, object-oriented ones, data flow diagrams, and sequence diagrams, contract-oriented analysis and design incorporate the following:

- ▶ *Legal model*: This specifies application requirements expressed

- in a set of unambiguous and machine-executable terms.
- ▶ **Logic model:** This expresses executable business logic during computations and the detection of conditions that trigger obligations of parties to a contract.
 - ▶ **Interactive model:** This emphasizes the flow of control and data between contracts or terms and external environments.

For a given service, developers (under the guidance of attorneys) use the legal model to describe requirements in the form of one or more

contracts and then extract the logical and interactive models. The models are used for software development, deployment, and execution. The legal model has a fundamental position in the legal-logic-interactive assemblage. To better understand this, we take an online car rental system as a use case to describe the common development procedure for a smart contract application. As documented in Figure 1, this system consists of the following four phases:

1. **Contract design:** A developer combs the online car rental

business process, designs a legal-logic-interactive model of a contract-oriented system, and writes the related contracts (as templates). The system is divided into four steps described in the boxes at the top of Figure 1.

2. **Contract development:** A developer refers to the business process and templates to write a smart (legal) contract. In Figure 1, we give a contract sample (including a description of parties, terms, assets, conclusions, and signatures) written

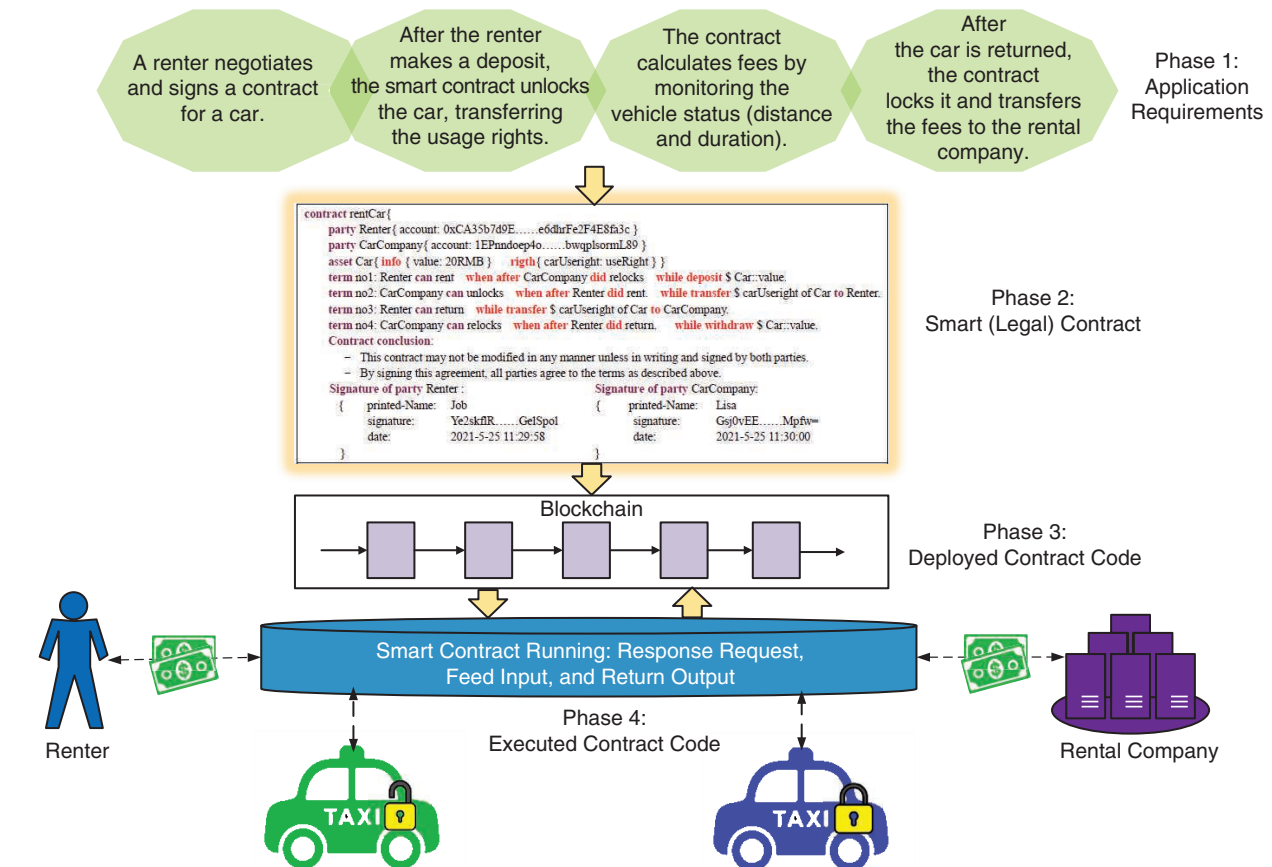


FIGURE 1. Smart contract engineering for an online car rental system.

- in Specification Language for Smart Contracts (SPESC).
3. *Contract deployment*: The smart contract is translated into executable code on a specified platform, and the code is deployed.
 4. *Contract execution*: A renter negotiates and signs the contract for a car and transfers funds to trigger the code execution to unlock the vehicle. The contract calculates fees by monitoring the vehicle's status (distance and duration) and, after the car is returned, locks it

and transfers the money to the rental company.

In this use case, once the smart contract system observes external events and account transactions, it executes the contract code according to pre-defined conditions.

BLOCKCHAIN SMART CONTRACT FRAMEWORK

Blockchain-based smart contracts are distributed frameworks to support contract-oriented programming through transaction-driven software

development. As shown in Figure 2, this framework can be divided into four layers, as in the following:

1. The contract layer supports the development and implementation of smart contracts derived from business scenarios in the form of text, executable scripts, and codes. It mainly relates to contract languages (for example, Solidity, Serpent, and Stack script) and trustworthy execution environments (such as a virtual

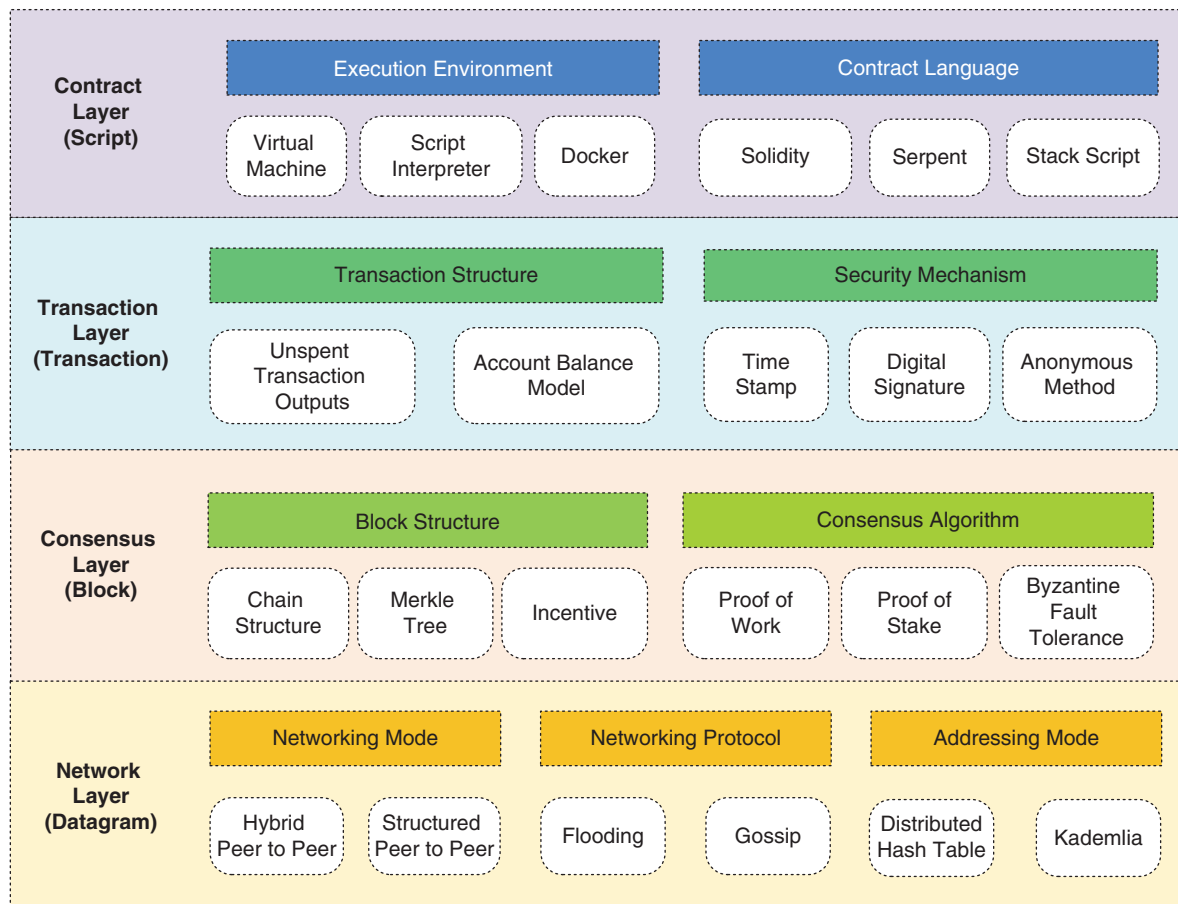


FIGURE 2. A framework for generating a general blockchain-based software architecture to support smart contract engineering.

machine, a script interpreter, and Docker).¹⁰

2. The transaction layer packages smart contracts and their execution results as JavaScript Object Notation (JSON) values into blockchain transactions, which are usually in a JSON form containing “key values.” It includes transaction structures (for instance, unexpended transaction output models and account balance models) and related security mechanisms (including time stamps, digital signatures, and anonymous methods of traders).¹¹
3. The consensus layer realizes the sharing of transactions in a blockchain network. A block is a formatted unit that bundles all authentic and valid transactions in a certain period of time, and consensus ensures that each block added to a chain is accepted by a majority of the nodes. This layer involves a block structure (for example, a singly linked list, Merkle tree, and incentive mechanism) and consensus algorithm (such as proof of work, proof of stake, and practical byzantine fault tolerance).
4. The network layer builds a peer-to-peer (P2P) system to support an efficient broadcast or multicast of a consensus layer. It usually views all transmitted data as datagrams and uses an overlay network architecture, including the networking mode (such as hybrid P2P and structured P2P), networking protocol (for instance, flooding and gossip), and addressing mode

(for example, distributed hash tables and Kademlia).

In addition, the application layer is set beyond the contract layer, and the existing TCP/IP protocol family is below the network layer. Each layer in the hierarchy treats everything it receives from the one above it as data and adds control information to ensure proper delivery. The addition of delivery information at every layer is called *encapsulation*. Moreover, each layer has its own data structure and terminology to describe it; for example, contract scripts are wrapped in a transaction, the transaction is packaged in a block, and the block is spread through blockchain networks in the form of a datagram.

A transaction is the basic unit of business processing in blockchain applications. It can establish complex associations with other transactions through hash pointers, which are cryptographic transaction hashes.¹² As opposed to traditional software architectures, hash pointers can be considered global unique identifiers and tamper-resistant evidence due to their 160/256-bit length and collision resistance properties. This uniqueness does not depend on a central registration authority or coordination between parties. Generally, the consensus layer and those below it are transparent to application developers.

SUPPORT FOR SMART CONTRACT ENGINEERING

Based on the preceding framework, a general software architecture for smart contracts is presented in Figure 3. This design inherently promotes contract-oriented programming, which is a paradigm that uses a “contract” as an alternative unit of computer work to

craft and implement integrated business applications and mission-critical software. On this basis, with the participation of lawyers, a traditional contract is written to express a business process.

The architecture in Figure 3 can treat an agreement as a smart contract as follows:

1. The smart contract development mechanism provides a programming framework, including a specified language, code editor, and compilation tools, to translate a traditional contract into executable code. This usually involves two steps: 1) the smart legal contract module employs domain-specific legal language to interpret the agreed operations in the traditional contract (its output is called a *smart legal contract*), and 2) the executable contract module generates the smart contract in the target language.
2. The smart contract deployment mechanism provides tools to position the executable contract in a blockchain, including 1) creating an instance of the contract and setting its variables to suitable values by means of negotiation between the parties to the agreement, 2) encapsulating the contract information (for example, the executable code, contract account, parties’ signatures, and incentive mechanism) into a blockchain transaction, and 3) incorporating the transaction in the blockchain through the consensus protocol.
3. The smart contract execution mechanism provides a trusted environment for fulfilling

the agreement, including 1) observing external events and internal transactions to trigger implementation, 2) building a

virtual machine (such as Docker and a sandbox) to run the code through the execution engine and instruction system, and

3) updating the contract state after predetermined conditions are met; the revised state is verified via the consensus protocol.

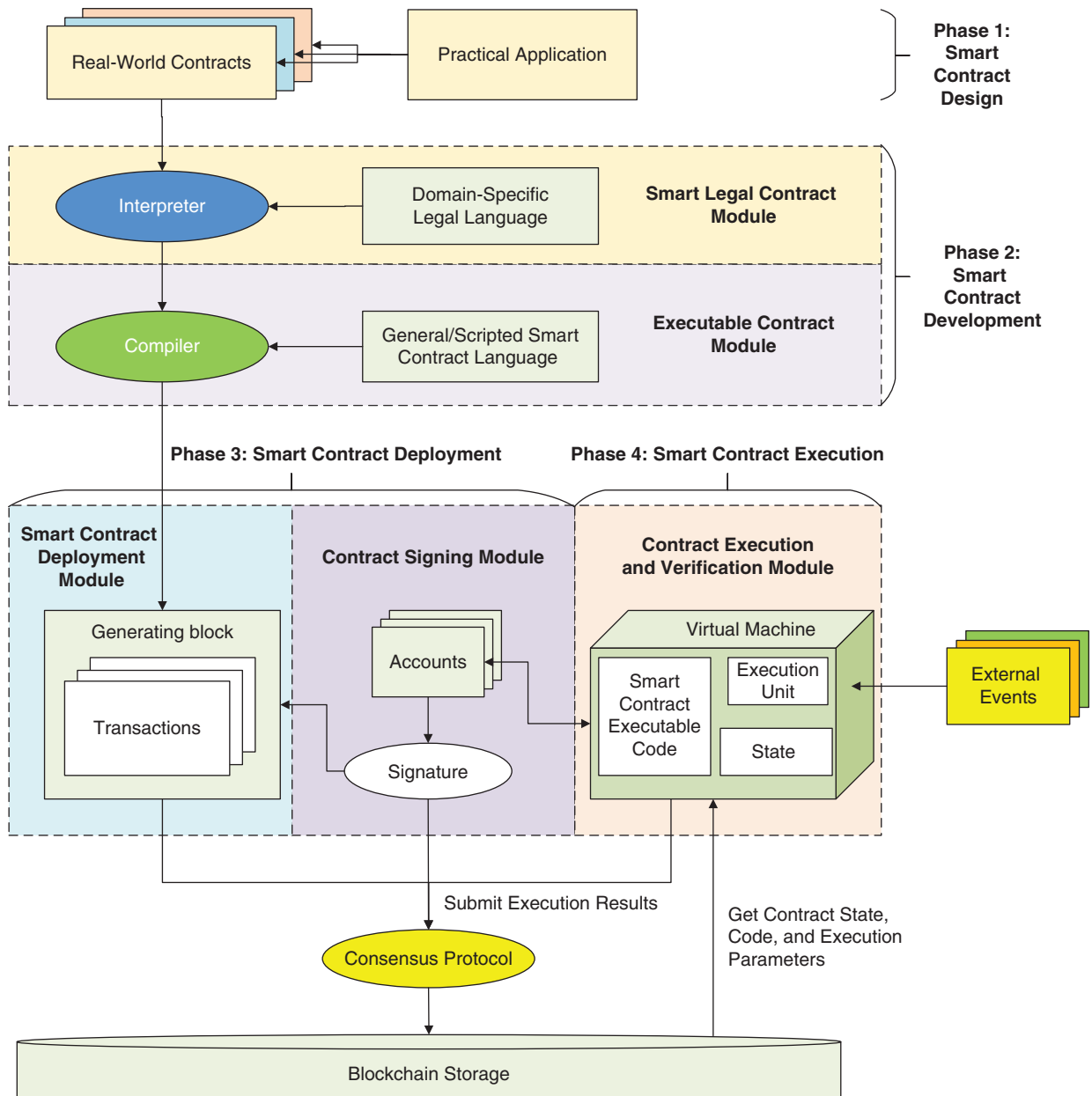


FIGURE 3. Support for smart contract engineering to develop the application in Figure 1.

From a business perspective, blockchain technology, as an innovative transaction network and a cryptocurrency, promises to transform smart contracts into a software payment mode. Thus, smart contracts have potential to produce a radically different competitive future for software services.

Smart contract languages supporting flexible service application development

A smart contract language is the intermediary between various services and platforms in practical applications, and it is an important tool to help users generate code quickly. Smart contract platforms may have a preferred language. For example, Bitcoin uses a Stack-based scripting language, and Ethereum supports two programming languages: Serpent (which is similar to Python) and Solidity (which is similar to JavaScript).¹³ Most other platforms facilitate smart contract development using general programming languages (such as C, C++, and Java). For instance, Hyperledger supports Go and Java, which interacts with blockchain through a specific interface.

According to the type of language, smart contracts can be divided into three categories:

1. *Scripted smart contracts*: These use a Forth-like Stack-based scripting language to realize elementary calculation, cryptographic primitives, and concatenative programming, for example, Bitcoin's scripting system.
2. *General smart contracts*: These employ a general programming language to develop Turing-complete smart contracts executed on a virtual machine or Docker. For instance, Neo can compile Java, Python, and other languages into a unified virtual machine instruction set.
3. *Specific smart contracts*: These harness a domain-specific language to tackle a particular problem, for instance, Contract Modeling Language, Digital Asset Modeling Language,¹⁴ and SPESC.¹⁵

Smart contracts are interdisciplinary, involving business, finance, law, and computers. Their design and development require close cooperation between experts in different fields. In recent years, smart legal contract languages, such as SPESC, have attracted extensive attention in academia. SPESC is based on the grammatical structure of a traditional contract and written in a form similar to natural language. It not only clearly defines the obligations and rights of parties but it includes the basic elements of a binding contract (including offers and acceptance, consideration, mutuality, and intentions). Hence, it can facilitate cooperation between legal experts and programmers.

Smart contract deployment with security and privacy

Security and privacy are primary concerns even for traditional contracts. However, blockchain is the perfect decentralized environment for smart contracts, as all data can be encrypted and put on an immutable ledger.¹⁶ This means contract information stored in blocks can never be lost, modified, and deleted. But this is far from all that blockchain can provide for smart contracts. In the following, we further explore the notions and approaches behind smart contract deployment.

When a contract is deployed in a blockchain, a separate account, known as a *contract account*, is opened to assemble all relevant costs and codes. Similarly, each contracting party has an account controlled by a private key, called an *individual account* or *wallet*. Any user can trigger the contract terms by sending a transaction from her wallet, and the code transfers her assets to a contract account or vice versa. A contract account, as an intermediary, has escrow authority to lock funds, which can be released under conditions acceptable to the parties. This approach can remain secure during online payment, clearing, and settlement. Smart contracts are digitally signed via the contracting parties' private keys and can be verified only by a public key stored previously in a blockchain. This is a simple, anonymous, and low-cost way to authenticate the identity of a signer without the need for a third-party public key infrastructure.¹⁷

Blockchain can be regarded as a decentralized, shared database. Blockchain smart contracts facilitate value exchanges and transactions to be carried out among anonymous parties without the need for an external enforcement and legal systems. Transactions are completely transparent, irreversible, and traceable, so anyone can review, audit, and validate them. In a word, smart contract platforms have shielded many technical details in blockchains so as to make various complex mechanisms (for example, signatures, consensus, mining, and so on) safe. Thus, programmers can focus on business processes by deploying tools provided by the platforms.

Trusted smart contract execution

When triggering conditions are satisfied, blockchain smart contracts

self-execute reliably and efficiently. In some cases, punishment will be imposed if terms are improperly carried out. This process, from practice experience, involves several important mechanisms. First, according to the contract runtime environment, there are the following three execution modes:

1. Script mode, first adopted by Bitcoin, executes code through a script interpreter, and it is limited to verifying the validity of a transaction. A transaction includes input and output scripts, which are used to unlock the output of a previous transaction and set the unlocking conditions for a transaction, respectively.
2. Virtual machine mode provides a completely transparent runtime environment to developers by abstracting resources from a physical machine. It shields differences among blockchain nodes to make execution consistent on all of them.
3. Docker mode enables developers to pack smart contract source code and its dependencies into a virtual container: a small and lightweight execution environment. It is more isolated and flexible and has more resources to call than virtual machine mode. Hyperledger Fabric is an example of Docker mode.

Second, monetary incentive is necessary. Contract execution in blockchain nodes results in consumption, so clients are required to prepay a certain amount (such as Ethereum gas). If a prepayment is too small to execute all operations, the processes will fail, and contract states will be rolled back. An


incentive mechanism can also prevent attacks on and abuses of resources. Further, smart contracts cannot access data from outside a blockchain network, so a trustworthy service, Oracle, is proposed to feed them external information (such as share prices, payment statuses, and credit ratings) that can trigger predefined actions. In addition, internal transaction states can be regarded as triggering conditions to activate the execution of a corresponding term in a recipient account. Finally, in the process of contract execution, transactions are used to store changing contract states, but this behavior must be agreed to by other nodes. This is known as the *all-or-nothing atomicity principle*, meaning that if a transaction wants to alter two values at the same time, both changes are made or neither of them is.

Smart contracts were conceived to automate value transfers and enable parties to agree to the outcome of an event without the need for trusted intermediaries. They are also a new software design and development architecture based on distributed ledgers and virtual technology. Due to blockchain immutability, decentralization, traceability, and anonymity, smart contracts are guaranteed to have the following four valuable characteristics:

1. *Programmability*: The terms of a contract are expressed as execution code based on programmed logic.
2. *Self-execution*: Coded terms will be automatically carried out as soon as predefined conditions are satisfied.
3. *Effective supervision*: Contract execution can be monitored at

runtime and verified by consensus, and full transaction records can be audited, thus reducing the need to actively collect, verify, and deliver data.

4. *Legalization*: Top-level contract design is based on legal constructs to ensure that execution complies with the parties' will.

In the near future, contract-oriented software architecture may help us to achieve high-performance application development. The emergence of smart legal contracts also demonstrates a significant improvement of source code comprehensibility, automatic programming levels, and software quality. The popularization of smart contracts will have far-reaching influence on services, judicial systems, and social governance and facilitate a new digital society with "low costs, high efficiency, and intelligence." To use blockchain to effectively develop service applications, much work needs to be done. One major challenge is to effectively build large-scale and complicated service applications, which require coordination among multiple teams.¹⁸ 

ACKNOWLEDGMENTS

This work was supported by the National Key Technologies R&D Programs of China (grant 2018YFB1402702), National Natural Science Foundation of China (grant 61972032), and European Union Horizon 2020 research and innovation program, under grant 952697 (ASSURED).

REFERENCES

1. S. Aggarwal and N. Kumar, "Blockchain 2.0: Smart contracts," *Adv. Comput.*, vol. 121, no. 2, pp. 301-322, Apr. 2021.
2. K. Wang, G. Lin, K. Kuo, H. Lee, B. Tsai, and W. Peng, "An empirical study of

ABOUT THE AUTHORS

YAN ZHU is a professor at the University of Science and Technology Beijing, Beijing, 100083, China. His research interests include cryptography, secure computation, and network security. Zhu received a Ph.D. from Harbin Engineering University, China. Contact him at zhuyan@ustb.edu.cn.

QIAN GUO is a postgraduate student at the University of Science and Technology Beijing, Beijing, 100083, China. Her research interests include blockchain techniques and smart contracts. Guo received her bachelor's from Zhongbei University, China. Contact her at guoqian@xs.ustb.edu.cn.

HONGJIAN YIN is a Ph.D. candidate at the University of Science and Technology Beijing, Beijing, 100083, China. His research interests include information security and cryptography. Yin received his master's in applied mathematics from Xidian University. Contact him at yinhongjian@xs.ustb.edu.cn.

KAITAI LIANG is an assistant professor at Delft University of Technology, Delft, 2628 XE, The Netherlands. His research interests include applied cryptography and information security. Liang received a Ph.D. from the Department of Computer Science, City University of Hong Kong. He is a Member of IEEE. Contact him at k.liang@surrey.ac.uk.

STEPHEN S. YAU is a professor of computer science and engineering at Arizona State University, Tempe, Arizona, 85287, USA. His research interests include services and cloud computing, cybersecurity, and software engineering. Yau received a Ph.D. in electrical engineering from the University of Illinois, Urbana. He served as president of the IEEE Computer Society and editor in chief of *Computer*. He is a Fellow of IEEE and the American Association for the Advancement of Science. Contact him at yau@asu.edu.

10. M. Coblenz, J. Sunshine, J. Aldrich, and B. A. Myers, "Smarter smart contract development tools," in *Proc. IEEE/ACM 2nd Int. Workshop on Emerg. Trends Softw. Eng. Blockchain (WETSEB)*, 2019, pp. 48–51.
11. M. Hamilton, "Blockchain distributed ledger technology: An introduction and focus on smart contracts," *J. Corporate Accounting Finance*, vol. 31, no. 2, pp. 7–12, 2020. doi: 10.1002/jcaf.22421.
12. M. Niranjanamurthy, B. Nithya, and S. Jagannatha, "Analysis of blockchain technology: Pros, cons and swot," *Cluster Comput.*, vol. 22, no. S6, pp. 14,743–14,757, 2019. doi: 10.1007/s10586-018-2387-5.
13. P. Cuccuru, "Beyond bitcoin: An early overview on smart contracts," *Int. J. Law Inform. Technol.*, vol. 25, no. 3, pp. 179–195, 2017. doi: 10.1093/ijlit/eax003.
14. M. Wöhrer and U. Zdun, "Domain specific language for smart contract development," in *Proc. IEEE Int. Conf. Blockchain Cryptocurrency (ICBC)*, 2020, pp. 1–9.
15. E. Chen et al., "SPESC-Translator: Towards automatically smart legal contract conversion for blockchain-based auction services," *IEEE Trans. Services Comput.*, early access, May 4, 2021.
16. S. Rouhani and R. Deters, "Security, performance, and applications of smart contracts: A systematic survey," *IEEE Access*, vol. 7, pp. 50,759–50,779, 2019. doi: 10.1109/ACCESS.2019.2911031.
17. M. Al-Bassam, "SCPki: A smart contract-based PKI and identity system," in *Proc. ACM Workshop on Blockchain, Cryptocurrencies Contracts*, 2017, pp. 35–40.
18. S. S. Yau and J. S. Patel, "Application of blockchain for trusted coordination in collaborative software development," in *Proc. IEEE Comput., Softw., Appl. Conf. (COMPSAC)*, 2020, pp. 1036–1040.

- an open ecosystem model for inclusive financial services," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, 2020, pp. 412–417.
3. N. Szabo, "Smart contracts: Building blocks for digital markets," *EXTROPY: J. Transhumanist Thought*, vol. 18, no. 2, 1996.
 4. T. M. Hewa, M. Ylianttila, and M. Liyanage, "Survey on blockchain based smart contracts: Applications, opportunities and challenges," *J. Netw. Comput. Appl.*, vol. 177, p. 102857, 2021. doi: 10.1016/j.jnca.2020.102857.
 5. V. Buterin et al., "A next-generation smart contract and decentralized application platform," *White Paper*, vol. 3, no. 37, 2014.
 6. A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, "HAWK: The

- blockchain model of cryptography and privacy-preserving smart contracts," in *Proc. IEEE Symp. Security Privacy*, 2016, pp. 839–858.
7. D. Drummer and D. Neumann, "Is code law? current legal and technical adoption issues and remedies for blockchain-enabled smart contracts," *J. Inf. Technol.*, vol. 35, no. 4, pp. 337–360, 2020. doi: 10.1177/0268396220924669.
 8. A. Savelyev, "Contract law 2.0: 'smart' contracts as the beginning of the end of classic contract law," *Inform. Commun. Technol. Law*, vol. 26, no. 2, pp. 116–134, 2017. doi: 10.1080/13600834.2017.1301036.
 9. M. Kolvart, M. Poola, and A. Rull, "Smart contracts," in *The Future of Law and Etechnologies*. Berlin: Springer-Verlag, 2016, pp. 133–147.