International Journal on Software Tools for Technology Transfer (2024) 26:431–444
https://doi.org/10.1007/s10009-024-00758-x

GENERAL

Special Section: Challenges of Software Verification

Software verification challenges in the blockchain ecosystem

Luca Olivieri[1] · **Fausto Spoto**[2]

Accepted: 27 June 2024 / Published online: 12 July 2024

Abstract
Blockchain technology has created a new software development context, with its own peculiarities, mainly due to the guarantees that the technology must satisfy, that is, *immutability*, *distributability*, and *decentralization* of data. Its rapid evolution over the last decade implied a lack of adequate verification tools, exposing developers and users to critical vulnerabilities and bugs. This paper clarifies the extent of block chain-oriented software (BoS), that goes well beyond smart contracts. Moreover, it provides an overview of the challenges related to software verification in the blockchain context, encompassing smart contracts, blockchain layers, cross-chain applications, and, more generally, BoS. This study aims to highlight the shortcomings of the state-of-art and of the state-of-practice of software verification in that context and identify, at the same time, new research directions.

Keywords Blockchain · Smart contracts · Blockchain-oriented software · Software verification · Program analysis · Automatic verification

1 Introduction

The first killer application of blockchain has been Bitcoin [1], in 2008. Since then, the technology evolved beyond its initial financial purposes and has been applied in heterogeneous ways and in different industrial and academic fields [2–6]. On one side, this rapid evolution facilitated the adoption of blockchain in various contexts. On the other side, development tools have not evolved as quickly, leaving open challenges, pitfalls, and space for improvement.

Software verification is among the top challenges for blockchain technology. Not surprisingly, it is highly needed and desired by blockchain developers [7, 8]. Namely, the history of blockchain is full of critical incidents, such as the DAO attack [9] (that allowed hackers to steal more than 50M USD) and the Parity wallet bugs [10] (that allowed hackers to freeze about 150M USD). These events have increased the awareness about software security in the blockchain community. Consequently, the latter is increasingly looking for precise and reliable tools and software layers to verify the software of the blockchain ecosystem in an automatic way.

This paper clarifies the extent of *blockchain software*, which encompasses much more than simply smart contracts, and provides an overview of the open challenges related to the verification of that software. It highlights problems and shortcomings of the state of the art and of the state of practice and proposes new directions for research.

Paper structure Section 2 provides an overview of blockchain technology. Section 3 discusses the kind of software involved in the blockchain ecosystem. Section 4 reports the issues related to bug fixing and patch management in the blockchain software context. Section 5 discusses the state of the art of verification tools for blockchain. Section 6 describes shortcomings and challenges of current blockchain verification. Section 7 discusses related work. Section 8 concludes.

✉ F. Spoto
fausto.spoto@univr.it

L. Olivieri
luca.olivieri@unive.it

[1] Ca'Foscari University of Venice, Venice, Italy

[2] University of Verona, Verona, Italy

2 Blockchain overview

A blockchain is an abstract shared data structure composed of a chain of blocks (see Fig. 1). Blocks contain a certain bounded amount of data records. When a new block is added, it is concatenated to the previous one, thus creating a chain of linked blocks. Typically, each block contains three fields:

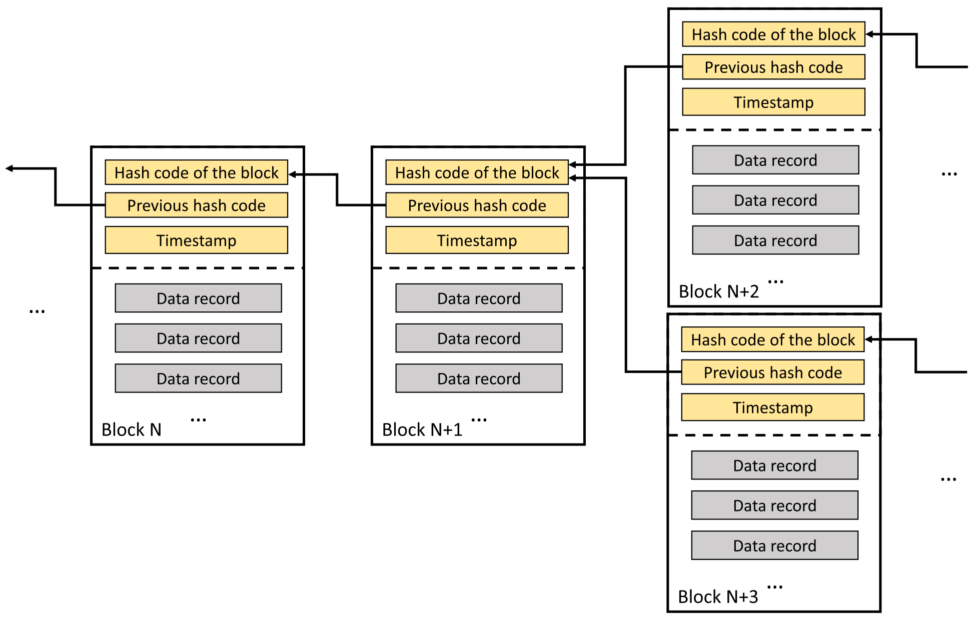

Fig. 1 High-level structure of a blockchain. It may contain forks, as shown here after block $N + 1$

1. The *hash code of the block*: an alphanumeric value with a fixed length that uniquely identifies the block. If any of the data related to the block is changed, then the block's hash code also changes.
2. The *previous hash code*: the hash code of the previous block.
3. A *timestamp*: a field that specifies when the block has been created and helps to maintain the chronological order of the chain.

These three components make the blockchain structure tamper-proof, that is, a modification of a block changes the hash and timestamp of that block. As a result, this leads to a mismatch with the hash codes stored in the blocks to which it is linked, making it immediately clear that the chain has been altered. The data structure is typically shared in a peer-to-peer network, the *blockchain network*, to achieve the following data properties [11]:

- *Distributability*: each peer of the network keeps a copy (full or partial) of the blockchain data and approves transaction requests to add new data through a consensus mechanism.
- *Decentralization*: the network peers are located in different geographic areas, avoiding single points of failure.
- *Immutability*: the antitampering properties, together with distributability and decentralization, make the data *immutable* (or hardly tamperable).

2.1 Blockchain network

Blockchain networks can be mainly classified into two different kinds:

1. *Permissionless blockchains* (or *public* blockchains) provide open networks, have no reference property or actor, and are designed not to be controlled and managed. The peers can join the network without previous authorizations and can be directly involved in the consensus and data validation process. Typically, the characterization of these networks is to have a high decentralization, full transparency of transactions, and no central authorities. This implies that these networks provide greater security in terms of points of failure and of consensus as it is difficult to corrupt most of the networks as they grow. Notable examples are Bitcoin [1, 12], Ethereum [13, 14], and Tezos [15, 16] blockchains.
2. *Permissioned blockchains* (or *corporate* or *private* blockchains) provide closed networks composed of known peers, such as members of a consortium, that interact and participate together or partially in consensus and data validation. Typically, decentralization is limited in the sense that it is distributed across a restricted number of parties, rather than across an unknown and potentially unlimited number of participants, as in permissionless blockchains. Also, in this case, there is no central authority, although there is a private decentralized group of users with network administration privileges. Notable examples are Hyperledger Fabric [17, 18] and Tendermint [19, 20].

Both network paradigms allow for similar value propositions, i.e., create a network of peers where it is possible to interact with the blockchain through transactions and with a certain degree of guarantees. However, their differences make them more suitable for some use cases and less suitable for others. Permissionless blockchains tend to be used in the contexts with a strong financial component or that require high degrees of decentralization, such as cryptocurrency exchanges, digital assets, crowdfunding, donations, and decentralized autonomous organizations. Instead, permissioned blockchains are favored for applications that depend on confidential data such as supply chain provenance tracking, claims settlement, and identity verification. As described in Sect. 4, the choice of the type of blockchain network has also implications on bug fixing and code patching.

2.2 Consensus mechanisms

In distributed contexts, it is common to find cryptographic infrastructure algorithms such as *PKI*s (*Public Key Infrastructures*) used for the secure exchange of information. In addition, for distributed data, it is fundamental to achieve consensus among network peers to avoid inconsistency and to decide which data can be validly added and which cannot be stored among peers. The main technological innovation brought by blockchain networks is the introduction of an incentive system that allows peers to act collectively to guarantee the integrity and security of the network. Blockchain is based on the principle of *trustless*, in which no one must necessarily trust third parties or individual peers. Trustless does not mean complete removal of trust, but rather its distribution in a type of economy that encourages certain behaviors and punishes others [21]. In this way, it adds a social component (i.e., unrelated to a computer algorithm but bound to human perception) that allows us to solve stalemates and conflicts related to trust. The consensus mechanism with rewards and disincentives is the backbone of blockchain technology because it ensures the validity and authenticity of the data stored in a blockchain. There are several fault-tolerant mechanisms [22, Ch. 11] that can be exploited by consensus-based systems, to reach a consensus on a single state of a network of distributed peers. Currently, the most popular paradigms for the blockchain context are *Proof-of-Work* (PoW) and *Proof-of-Stake* (PoS). Specifically, PoW requires actors called *miners* to solve computationally expensive puzzles to validate transactions and create new blocks. The action to solve the puzzle is called *mining*. The first miner that solves the puzzle gets the right to add a next block to the blockchain and receives a reward. The main disadvantage is that PoW consumes a lot of energy due to the computing power required for mining.

PoS, on the other hand, does not involve miners but rather actors called *block validators*. PoS selects validators based on the amount of *stake* they reserved as collateral. The amount of stake is typically composed of economic assets such as cryptocurrency. If a validator behaves unfairly, then part or all of the amount will be deducted. PoS is more energy-efficient than PoW since it does not require extensive computational power, but relies instead on the economic incentives for validators, to maintain network fairness.

3 Software in the blockchain ecosystem

Before dealing with software verification, it is important to understand the type of software present in the blockchain ecosystem and its extent. Blockchain can be thought of as an abstract data structure shared in a complex ecosystem, where software allows us to build the system and implement interactions with the components that make up the ecosystem. The term *blockchain software* is widely used, but it has different meanings. The first that comes to mind is *smart contracts*, i.e., programmable executable code within the blockchain. However, this is only a small part of what might correctly be named as blockchain software. For this reason, it is necessary to define from the very beginning the various types of software involved in blockchain technology.

Figure 2 provides an overview of the blockchain ecosystem. In this context, the software can be categorized into *blockchain software* and *blockchain-oriented software* (BoS). As expressed by their names, both categories involve blockchain technology, but they differ on where the software is located and executed, and they have different purposes.

3.1 Blockchain software

The definition of *blockchain software* includes all code present within the blockchain and its network, i.e., the implementation code of the blockchain itself and that contained in the database of blocks. According to Marijan et al. [23], even though there is not always a clear distinction, the code related to the blockchain implementation can generally be divided into software layers (see Table 1). It is similar to the ISO/OSI model [24], but it contains some domain-specific layers such as the *Consensus* and *Data* layers, which handle and implement the consensus mechanism and data storage and management, respectively.

Smart contracts are also considered as part of the blockchain software. They are programs that can be immutably deployed and executed within the blockchain, whose codes are stored in the data layer, and whose execution is performed by a smart contract framework located in the application layer. Their original meaning of agreement between parties [14, Ch. 7] is nowadays blurred, given the genericity of the software that runs within modern blockchains, especially after the adoption of Turing-complete languages for smart contracts, as in Ethereum [14].

Cross-chain communications allow us to perform transactions across different blockchain networks seamlessly and without centralized intermediaries. Cross-chain communication protocols are part of blockchain software as they must be integrated within the blockchain. However, since there are no standards but only limited solutions [25], it is not always clear at which layer they are typically implemented.

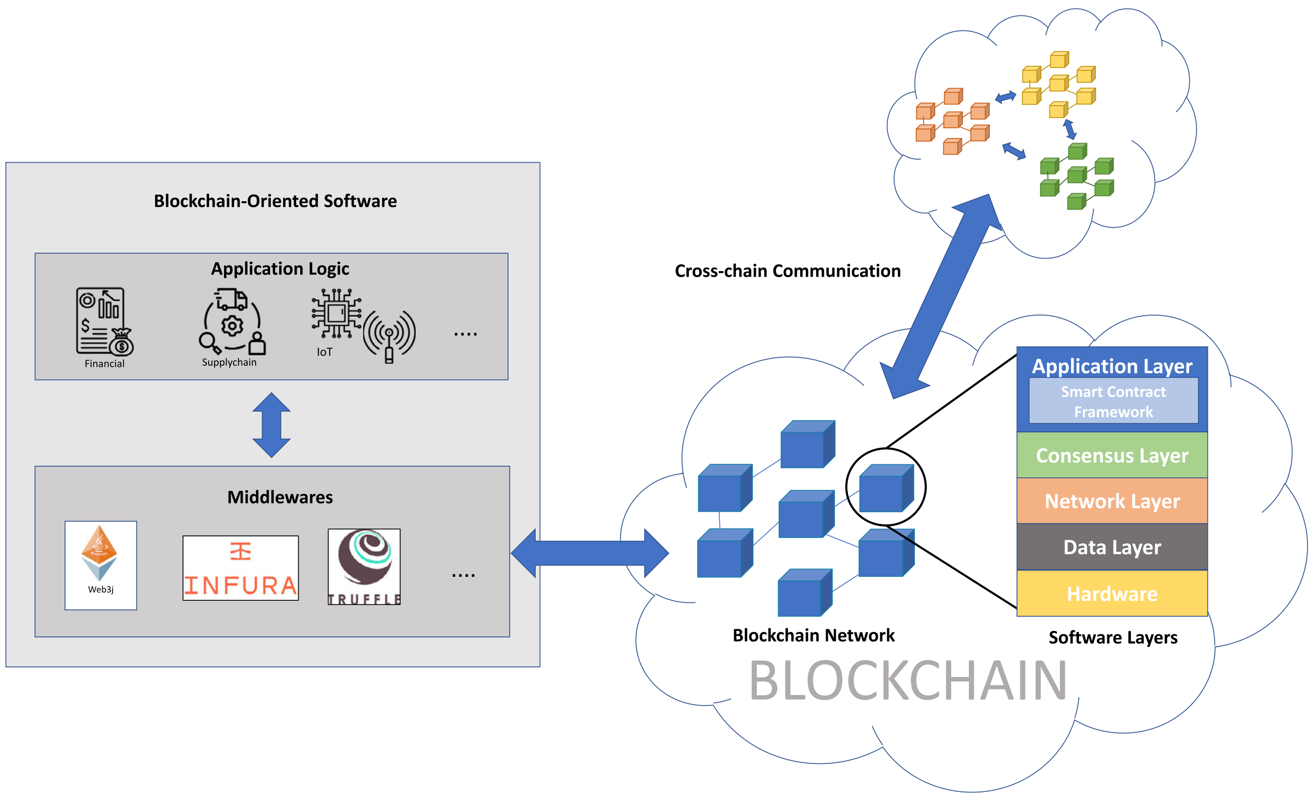

Fig. 2 Software classification in the blockchain ecosystem

Table 1 Layer division of blockchain software

Layer	Description
Application	It contains the code that manages the content of transactions, proposes updates to the database of blocks with new data, and performs additional operations.
Consensus	It implements the logic of the chosen consensus mechanism to validate or reject the data to store in a blockchain in agreement with the blockchain network.
Network	It deals with communication in the blockchain network, such as P2P protocols. It allows us to connect the various peers, handle transactions, and propagate information across the network.
Data	It contains the implementation of a database of blocks with its primitives. It defines the data and block structure, how data gets added, and any additional information included in the data.
Hardware	It contains the software to work at low level within the hardware of the devices. The blockchain network consists of different devices supported by heterogeneous hardware. Typically, the code of this layer can be involved to virtualize the hardware and make the blockchain software platform-independent. Otherwise, it is exploited to optimize the performance of an ad hoc hardware architecture such as in Internet-of-Things devices.

3.2 Blockchain-oriented software

The definition of *blockchain-oriented software* (or *blockchain-based software*), according to Porru et al. [7], includes all software working with a blockchain implementation, that is, software that interacts, directly or indirectly, with the blockchain but is located and executed outside the blockchain. Typically, blockchain-oriented software separates the application logic from the blockchain communication logic that is handled by middleware frameworks (e.g., Web3j, Infura, Truffle, . . .). Here, for example, the application logic can range from generic applications that use blockchain only as tamper-proof data storage (such as supply chains and IoT applications) to applications that actively interact with smart contracts (such as wallets, crypto-currency, and asset exchangers).

Finally, *Decentralized Applications* (DApps) are applications executed by multiple users over a decentralized network, such as a blockchain network. This definition broadly includes both blockchain software and the BoS definitions.

Indeed, users are not necessarily peers of the blockchain network. In general, decentralized applications have an external interface that can be used to communicate and receive information. For instance, in blockchains, popular DApps are decentralized autonomous organizations [26] and decentralized games [27, 28].

4 Bug detection, fixing, and patch management in blockchains

Developers of traditional software apply techniques such as continuous integration [29], where they integrate code frequently. Each integration is checked by an automated architecture that detects bugs and vulnerabilities as early as possible. This leads to a significant reduction of issues in production and allows the development team to build cohesive software more rapidly. This is true for blockchain software as well, with the limitation that blockchains are by nature distributed and decentralized [30, 31]. Consequently, the continuous integration tool cannot simulate a distributed execution context and all the problems that might arise in a real decentralized scenario, which will remain untested.

For *permissionless* blockchains, bug fixing and code integrations are rather hard. Namely, there are no network administrators in this scenario, participants do not trust each other and are often not even required to be authenticated. In general, any change has to be made on-chain through the consensus mechanism, that is, only after reaching an agreement of the network majority. This mechanism has several drawbacks. For instance, bugs affect smart contracts, which are typically *immutable* data within the blockchain and cannot consequently be fixed. The network majority would agree on rolling back to a previous safe state to apply the patch, effectively rewriting the history of the blockchain, which is against the idea of blockchain. However, the more the popularity of a blockchain network increases, the trickier it is to undertake such a turnaround. Part of the network peers may decide to ignore the change and continue as if nothing happened, leading to a new, independent blockchain, also known as a *hard fork*. Note that the rollbacks of changes and hard forks are incredibly rare events in the blockchain. For instance, currenly, that happened only once in a popular blockchain Ethereum because of the DAO attack [9], which had large-scale effects.

Instead, if bugs affect vital blockchain components such as the consensus layer of the blockchain, then they can lead to critical consequences such as the partial or full denial of service of the network, for instance, because of the introduction of nondeterminism [32, 33]. In this worst-case scenario, that compromises the consensus layer, no decision can be made on-chain, and nobody can use the blockchain properly because all transactions fail or the blockchain splits as in a hard fork. Therefore it is inevitably necessary to make off-chain decisions that increase the time and cost of bug fixing.

For *permissioned* blockchains, it is possible to patch buggy code through network governance. Typically, a limited subset of peers has the power to propose a plan for halting, modifying, and restarting the blockchain with updated software, carefully migrating the state to the previous version. This kind of solution is often adopted in the industrial field, especially for enterprise or consortium blockchains. However, enforcing an update requires to stop all services and data management. For instance, a blockchain such as Cosmos [34] offers an automatic process to deal with these problems and applies blockchain upgrades, improving the synergy between the on-chain module `upgrade`, responsible for halting the chain, and the off-chain daemon that installs a new binary of the node software at the right time and autonomously restarts the node. In this way, it is possible to achieve greater control of the network, improve blockchain performance, and apply patches and fixes quicker than in a permissionless blockchain.

For BoS, the software runs off the blockchain, so it can generally be patched as it is done for traditional software. However, the fact that it is patchable does not mean that it cannot send wrong or corrupted data to the blockchain, which stores it immutably, leading to subsequent problems during data retrieval or blockchain software execution. A typical example of BoS is a supply chain management system that records data on a blockchain. In this scenario the application/bussiness logic is often developed according to traditional programming paradigms as it is not located within the blockchain, whereas only the data produced by it or small portions of code are recorded within the blockchain. In this way, it is possible to improve and extend the software outside the blockchain. However, if it is affected by bugs or it inserts incorrect, wrong, or corrupted data, then it would be necessary not only patch the code outside, but also to fix the data within the blockchain with the issues of the case.

For cross-chain communication, developers must account for the issues of each connected blockchain networks, each falling into one of the previous scenarios. This makes bug fixing even more difficult in case of multiple permissionless blockchains.

5 State of the art of blockchain software verification tools

Several surveys [23, 35–37] show the state of the art and practice related to blockchain software verification tools. Blockchain and smart contract verification is more similar to that of critical software [38] than to the verification of traditional, desktop software. The main reason is that the blockchain, beyond its unique data properties (antitampering, decentralization, distribution), operates in a *trustless environment*, where peers do not trust each other. Consequently, code fixing may not be always possible or might be difficult to apply. Therefore verification tools must also highlight properties traditionally associated with quality rather than with safety, as these can have critical implications. In short, quality also means security in the blockchain context.

Automatic verification techniques can be divided into dynamic and static.

According to Chakraborty et al. [39], the most used dynamic technique is testing. In general, it executes the software and checks whether the execution specifically meets the expected results. In this way, although it does not exactly indicate the cause of the issue, it is possible to detect the presence of bugs, highlighted by wrong results. Most programming languages for smart contracts have explicit primitives for dynamic verification of invariants, although there is still a lack of general frameworks for testing smart contracts. In the case of other blockchain software, the testing frameworks for general purpose languages are actively used, such as JUnit for Java. However, according to Destefanis et al. [10], testing is challenging in the blockchain context. Namely, its application to a real-world blockchain network such as main-nets and public test-nets takes relatively long to run and deploy the code, exposes the code to possible malicious users, and can be financially expensive in terms of transaction fees and/or currency consumptions for execution and deployment of contracts. Instead, testing on local and private blockchain networks is generally fast and cheap in terms of fees and real currency used but has limited interaction and is not necessarily a realistic scenario. Moreover, testing has other drawbacks, not strictly related to blockchain technology: the creation and maintenance of test cases is not trivial and may require a big effort to define the requirements that must be covered and to compute the expected results for each input case. In addition, testing can only show the presence of bugs but never their absence [40, Ch. 3] because it can only observe a finite set of finite program executions [41, Ch.1.4.1].

Static techniques analyze program behaviors and program components before they are run. Typically, static verification tools trigger alerts to highlight program components of interest such as instructions with potential errors, bugs, and known security vulnerabilities. In addition, static verification reduces the cost of bug fixing for developers and gives them the chance to fix bugs and code smells at an earlier stage [41]. Furthermore, static analysis can be combined with formal methods to analyze software by exploiting mathematical theories and by ensuring the presence or absence of certain code properties, bugs, and vulnerabilities. Notable examples are abstract interpretation [40], model checking [42], and theorem proving [43].

Deductive verification techniques can also be applied to prove the correctness of the system [44]. However, this approach is not fully automated and typically requires the initial construction of the proof rules by hand.

Currently, the scenario of verification tools for blockchain software is mainly focused on smart contracts. In recent years, smart contract verification has seen a rapid growth: it started as mainly syntactic checks and evolved into modeling and checking program behaviors by using formal methods [45]. Moreover, several tools [46–48] use formal ways to represent verification conditions such as Constrained Horn Clauses (CHCs) [49]. In particular, the popularity of CHCs is due to the possibility of solving clauses efficiently by using Satisfiability Modulo Theories (SMT) [50] solvers and of quickly ascertaining the security of the smart contract, i.e., whether the CHCs are satisfiable and the property under verification is held. However, the main pitfall of this approach is that, in general, CHC satisfiability is undecidable. Therefore the analysis execution may not produce a result in some cases.

Regarding the other part of the blockchain software, i.e., the one not strictly related to smart contract frameworks, it lacks adequate tools [32], in general. Hence it is necessary to expand the range of tools that identify nontrivial errors with semantical analysis of the programming languages and with a formal guarantee that all errors of some category are found (*soundness*). However, as stated by Ferrara et al. [51], the development of formal verification tools based on formal methods requires a significant theoretical background and programming skills, even to design and implement a toy tool, making it difficult for traditional developers and blockchain practitioners.

6 Challenges, opportunities, and new directions

The evolution and progress of blockchain technology go toward a progressive increase in the complexity of the blockchain ecosystem. As a consequence, ensuring the high quality and security of software becomes more challenging every day: it is essential to design automatic verification systems and use appropriate tools. However, the rapid evolution of technology does not always allow for the development of adequate architecture and tools. Moreover, according to Marijan et al. [23], blockchain-based software development is still an emerging research discipline; therefore the best practices and tools for software verification are still underdeveloped. In the following, we discuss several topics in the blockchain verification context.

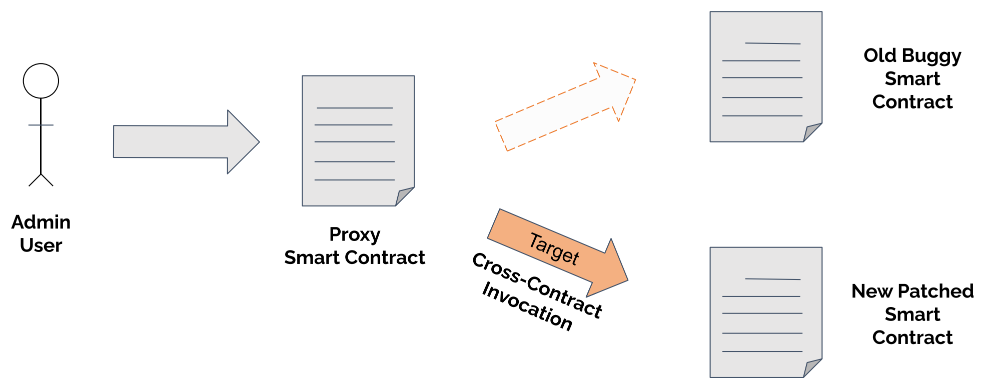

Fig. 3 Schema of the proxy upgrade pattern

6.1 Automatic verification architectures

Software verification of a blockchain ecosystem is nontrivial. It has all the difficulties of distributed and decentralized systems, with the typical addition of code immutability. Given these issues, it is of utmost importance to apply automatic software verification in the development process to identify critical bugs that can be exploited by malicious actors and compromise the blockchain network.

According to Mahdi et al. [30], traditional software paradigms, such as Software Development Life Cycle (SDLC), are inadequate for blockchain software development due to the problems related to bug fixing and patch management (see Sect. 4), leading some industrial companies to adopt inadequate architectures and tools [32]. This leads to two critical consequences: developers may spend time investigating bugs that do not exist, losing their faith in the tool suite, with the risk of removing safe code due to false positive results; moreover, the verification tools that do not properly model or support the code can lead to false negatives, leaving bugs and vulnerability in production and giving developers a false sense of safety and exposing the blockchain to high risks.

For this reason, new paradigms and solutions are emerging. For instance, Marchesini et al. [52] describe a software development process called *Agile Block Chain Dapp Engineering* (ABCDE) to gather the requirements and to analyze, design, develop, test, and deploy blockchain-oriented software. ABCDE complements the incremental and iterative development through boxed iterations, typical of agility, with more formal tools. Besides modeling interactions among blockchain-oriented software using UML, it also provides practices, patterns, and checklists to promote and evaluate the security of a DApp written in Solidity [53]. Furthermore, ABCDE was also applied in DApp development for enterprise purposes using Hyperledger Fabric [54]. These techniques are related to continuous integration, which allows us to build and deploy code only if it passes all compilation and testing requirements.

6.2 Smart contract bug fixing

A key point of blockchain-based software is that data in blockchain are immutable. Even though smart contracts are typically immutable data within the blockchain, new bugs and vulnerabilities are discovered every day. To overcome this issue, new research and design directions are being explored. However, they are still in an early stage, and like all new solutions, they have several drawbacks.

For instance, in permissionless blockchains, such as Ethereum, ERC-2535 [55] and the Proxy Upgrade Pattern [56] provide a way to extend and upgrade smart contracts after deployment. The basic idea is that the code of a single contract is typically immutable over time, whereas this is not the case about its state, which can change each time it is executed. By exploiting this idea it is possible to create contracts that dynamically target other contracts through cross-contract invocations. In this way, target contracts can be patched over time with the possibility of fully replacing their code (see Fig. 3). These solutions allow us to patch the code but have some drawbacks:

- *Higher costs*. Splitting a single contract into multiple subcontracts is more expensive in terms of gas [14], transaction, and deployment fees.
- *Trust in a third party*. To replace a buggy contract with a new version, we need to change the target. This operation cannot be done by anyone, since an attacker could otherwise target a malicious contract, leading to catastrophic consequences. For this reason, it is necessary to programmatically create an admin user contract that manages these changes. However, this means trusting a third party, which is not always possible in the blockchain, and which in some ways goes also against the *trustless* principle of the blockchain.
- *Unavoidable attacks*. While it is possible to apply post-deploy remediation patches by leveraging these paradigms, bugs, and vulnerabilities must first be detected. Consequently, although patchable, the contracts are exposed to attackers without timely detection.

- *Unexpected Behaviors*. The changes after the code update can have a significant impact on contract behaviors. If users do not notice these changes, then they may incur into critical compatibility issues.

Instead, in some permissioned blockchains, such as Hyperledger Fabric, smart contracts are not always immutable, but only the transactions that interact with them are. Hence these blockchains provide a process for upgrading smart contract code [57]. In this way, target contracts and blockchain global state can be patched over time, while the transaction and event log remains immutable. However, this presents the same problems as in the previous case and also typically requires governance to manage permission policies to execute the upgrate process.

Finally, [58] investigates an example of automatic verification over time. It proposes an *on-chain* code verification approach, where the blockchain verifies the code upon deployment. In this way, it is possible to make software verification mandatory, being performed directly by the blockchain, and avoid untrusted smart contract executions. This architecture allows the same blockchain to reject the code that does not pass a set of checks. Therefore verification becomes part of the consensus mechanism to ensure that all network nodes have reached the same verification result. A lazy reverification approach is also proposed to recheck the code already deployed before its execution, whenever the verification checks are updated, to provide an automatic verification over time. However, this kind of solution suffers from some issues:

- *Resource consumption*. Performing verification directly on the blockchain can slow down the blockchain network and burn computational resources. Typically, verification tools that apply formal methods to detect semantic properties may be computationally expensive, sometimes without affecting the precision of the analysis [59]. Therefore they must be adequately designed and optimized for efficiency to be applied in an on-chain verification architecture.
- *Freezing of smart contracts*. Reverification might deny the execution of bugged, already deployed contracts. This leads to freezing smart contracts, which requires some unblocking and patching mechanisms. Moreover, freezing smart contracts also freezes the funds held in those contracts. Hence it is also necessary to create nontrivial withdrawal strategies.
- *Denial of network service*. A lazy approach relies on on-demand reverification for scaling. However, this might not be the best choice. Theoretically, an attacker can send many requests to unverified contracts and create a denial of service of the network due to too many verification requests that degrade the network performance.

6.3 Verification tools

In addition to ensuring automatic verification, it is also necessary to design suitable verification tools to guarantee the quality and security of the code. This is not easy, since blockchain technology increases in complexity over time and the adoption of Turing-complete languages introduces the risk of all sorts of bugs [9, 60, 61].

In the following, we highlight several shortcomings of the state-of-the-art verification tools.

6.3.1 Cross-component issues

In the latest years, there has been significant development of verification tools for verifying smart contracts [62]. Currently, the majority of verification tools for smart contracts target individual contracts only. However, the most critical and challenging issues to detect are those that occur through the interaction of multiple contracts. Notorious and harmful attacks on smart contracts are related to cross-contract invocations such as reentrancy, parity wallet bug, and untrusted cross-contract invocations. Specifically, the reentrancy attack [60] happens because a smart contract calls a method in other smart contracts, assuming a specific, standard implementation of that method. However, malicious users can redefine the method to execute sensitive code on behalf of the caller, leading in this specific case to the repeated (*reentrant*) execution of money transfers. The parity wallet bug [10] implies the accidental alteration of smart contract libraries deployed in blockchain, making the funds of smart contracts inaccessible, depending on that library. The first case of parity wallet bug was accidentally triggered by an Ethereum user who deleted a smart contract library, resulting in the freezing of approximately $150 million worth of Ethereum cryptocurrency. Instead, untrusted cross-contract invocations [63, 64] occur when a contract can invoke a function from another contract, but in the code the target contract to be called is not hardcoded and can be modified by user inputs. In this case, a malicious user could change the target by redirecting the call to a malicious contract that can improperly retain economic assets or execute arbitrary code and returning arbitrary values. It is a more general scenario than reentrancy, which can be seen as a particular case of this problem.

In general, all these problems arise because smart contracts installed in a blockchain are exposed to an open environment, where users can dynamically install new smart contracts and make them interact. Therefore programmers must write very defensive code, since very little can be assumed about the context where their code will be executed. This is completely different from the case of traditional applications, which are meant to be complete software components and can be analyzed as such. The situation is similar to the development of a library that must integrate inside still unknown environments with the difference that smart contracts carry money and their failure might imply the lost of those funds.

As reported in Sect. 3, smart contracts are just the tip of the iceberg of the blockchain ecosystem. Therefore it is also necessary to consider problems that arise from the interactions of other components. For instance, if we consider a cross-layer, then an interesting issue is that of nondeterminism [32, 33]. Typically, issues of nondeterminism affect the consensus layer and happen when despite executing the same transactions, the result of the operations is different in the actors involved in the validation, leading to failed consensus and the consequent failure of the transactions. This may be due to various factors, including the execution of smart contracts at the application level containing functions that can return different values depending on who executes the code (such as random value generators or system calls) and that therefore lead to different blockchain states. In the worst cases, this can lead to the loss of money due to the failure of transactions in the case of transaction fees or to the denial of service of contracts or parts of them due to failure to reach consensus.

Other cross-component issues, such as numerical overflows, concern the whole ecosystem, spanning from the BoS [65] up to smart contracts [66, 67] and cross-chain applications [68].

6.3.2 Multilanguage issues

The blockchain ecosystem is heterogeneous in terms of software. Various components can be implemented in different programming languages, ranging from general-purpose languages (such as C++, Go, and Java) to domain-specific languages (such as Solidity, BitcoinScript, and Michelson). According to Negrini et al. [69], the software of the blockchain ecosystem is increasingly divided into separate subprograms, interacting with each other and choosing the best language for the task at hand. In terms of verification, this adds another layer of complexity to the analysis.

Namely, from a program analysis perspective, the use of many languages is challenging because there is a lack of techniques that work with different languages at the same time [70]. Moreover, the adoption of general-purpose languages implies extensive use of third-party libraries and frameworks, representing a challenge for verification, since customers expect the analysis to be aware of such libraries, whereas the effort of modeling even a single framework is high [71].

In general, although it is nontrivial to manage the verification of multilanguage code as regards the development of smart contracts, the problem of multilanguage within a single blockchain is often mitigated. Indeed, there are several blockchains that support frameworks for the development of smart contracts exploiting the *metaprogramming* paradigm [72]. Metaprogramming means developing smart contracts in different high-level languages that all compile to a single, normally low-level target language. In this way, it is possible to switch among popular high-level languages based on the programmer's preference and project requirements, keeping the low-level code compatible. Some notable examples are the languages Solidity and Vyper, which can be compiled into EVM bytecode for Ethereum [14] and IoTA [73, 74]; Pythonic and TypeScript-like languages provided by Archetype [75], LIGO [76], and SmartPy [77], which compile into Michelson [78] for Tezos; and other high-level general-purpose languages supporting WebAssembly [79] compilation for blockchains such as Cosmos [80], Polkadot [81, 82], and IoTA [83].

In terms of verification, this implies that it is sufficient to analyze the low-level code only, rather than necessarily supporting all high-level language variants. According to Olivieri et al. [64], smart contract verification at low level provides different advantages, compared with high-level languages: (i) it is more faithful, as it analyzes the code actually executed (or closer to), (ii) it enables the analysis of code when source code is not available (for instance, for smart contracts already deployed in blockchain), (iii) it avoids redundant work that the compiler has already performed, such as name resolution, type checking, template/generics instantiation, and (iv) it leverages the transformation of high-level constructs into low-level code, already performed by the compiler.

However, the verification of low-level languages carries several pitfalls with it, mainly due to the loss of information, making it difficult to understand, reverse engineer, analyze, and inspect the code. Typically, high-level languages feature compact instructions, types, and annotations. Instead, low-level languages have a restricted instruction set and make all operations performed during the execution explicit, losing expressiveness and increasing code verbosity. In addition, compilation problems may occur when the semantics of some high-level instruction may not be easily expressed in terms of low-level instructions (see Olivieri et al. [64] for examples in Tezos).

6.3.3 Lack of formal definitions, notions, and proofs

Formalization allows us to rigorously define the problems and vulnerabilities of the blockchain. This allows tools that use formal methods to automatically identify them in programs, helping developers to avoid any error or inconsistency that can lead to critical problems. However, verifying correctness is a nontrivial task, and sometimes properties are taken for granted without thorough verification and formal proofs. Consider, for instance, the idea of gas for smart contracts, as introduced by Ethereum [14]. When a smart contract is executed, it starts with an amount of gas consumed during execution. Depending on the gas cost model, the execution of some instructions burns gas. If gas is over before the end of the execution, then the latter will be aborted. The concept of gas is extremely important in smart contract execution because it copes with the risk of nontermination. However, it took some time before in-depth studies proved the correctness of the most popular gas models. According to Genet et al. [84], a gas model works only if it is formally guaranteed that the gas is consumed in every situation where nontermination may occur. As reported by the authors, this was not immediately established from the specification given by the Ethereum Yellow Paper [85], and only later it was proved that no program can execute indefinitely without consuming gas in the Ethereum execution model.

6.3.4 Lack of coverage in the layers

Currently, tools and studies regarding blockchain verification are mainly focused on the application layer. However, also the other blockchain layers contain software that is required to be verified and whose properties must be proven. Regarding these, the state-of-the-art presents a strong deficiency, not to mention the verification of interlayer interactions which is almost nonexistent. For instance, the consensus layer is of primary importance. It is one of the pillars of blockchain because it allows us to have networks where participants do not necessarily have to trust each other. However, vulnerabilities or bugs in their code could critically compromise the entire stability of the blockchain ecosystem. As far as we know, only a few works regarding consensus protocols are present in the scientific literature. Specifically, Kiayias et al. [86] provides a noninteractive theorem prover for PoW consensus like Bitcoin and Ethereum blockchains. Maung et al. [87] describe a formal approach based on model checking for the verification of the PoS consensus of Tendermint. Also, Yoo et al. [88] deal with the verification of the consensus protocol of Stellar [89] through model checking. Finally, Kawahara [90] describes a solution based on an SMT solver for the consensus protocols of Hyperledger Fabric.

Every day new blockchains are created, and potentially each of them can implement consensus protocols in a different way, which requires careful verification. The same principle can be applied to the other layers that manage data and information with different data structures, algorithms, and code implementations.

6.3.5 Compliance with laws and regulations

A rapid evolution of technological innovation does not always go hand to hand with legislation and regulations provided by various governments and countries. In this regard, blockchain technology is no exception. Although its first appearance with Bitcoin dates back to more than a decade ago, only recently are steps taken to regulate these technologies in countries such as those of the European Union (EU) despite that to apply blockchain technology in certain contexts, implementations must comply with laws and regulations.

For instance, the recent EU Data Act [91] forces to provide details on how nonpersonal data, metadata, and personal data are managed (including GDPR principles [92]) and specifies how smart contracts must be regulated for data sharing agreements. In particular, it sets essential requirements that smart contracts must comply with, whose satisfaction poses nontrivial challenges [93]. Haque et al. [94] provide a systematic literature review regarding blockchain and solutions compliant with personal data regulations. Their finding indicates that studies about these topics have increased in number. In particular, data deletion and modification seems to be blockchain's most discussed compliance issue, also highlighting that IoT and healthcare domains are the most discussed research areas. Instead, Molina et al. [95] design principles for GDPR-compliant blockchain solutions, identifying and discussing the challenges of GDPR requirements.

To the best of our knowledge, only Tauqeer et al. [96] dealt with program verification of such issues, proposing a solution based on knowledge graphs and semantically modeled informed consent [97] for GDPR compliance of smart contracts. However, other existing tools and techniques based on abstract interpretation for traditional software are present in the literature and could be adapted to the blockchain context [98–100].

7 Related work

Most of the literature about challenges and new research directions for the blockchain ecosystem is about smart contracts [101–106]. In these works, discussions related to software verification are often marginal or absent in favor of other design and development aspects.

Regarding software verification challenges, the literature is poor and presents only a few papers. Singh et al. [107] and Krichen et al. [108] propose reviews, where they present and analyze the state of the art concerning the formalization of smart contracts. However, they aim to highlighting the popularity, pros, and cons of each kind of formal methods. Differently, Marijan et al. [109] provide a more detailed and exhaustive analysis, including the different layers of blockchain software. Li et al. [110] and Islam et al. [111] describe common blockchain security issues but without highlighting challenges concerning verification tools and architectures. Magazzeni et al. [112] explore validation and verification challenges, focusing on legal smart contracts and natural languages. Koul [113] proposes an overview of challenges, strictly related to blockchain testing but without describing implementations based on formal methods.

8 Conclusions

Blockchain technology has created a new programming context, which encompasses several technologies such as cryptography, distributed systems, and programming languages. Although formal verification of software has a long history in computer science, its application to the blockchain context leads to new research and implementation challenges due to the lack of fundamental theory, immature verification architectures and tools, and missing standards. However, these issues have attracted the attention of academia, companies, and governments, which keep investing in their investigation. The evolution of blockchain technology in the coming years will inevitably affect the information sector, which is bound to bring new changes to our lives. For this reason, it is necessary to work in different directions, seeking the best solutions to ensure safety of this technology.

Funding Open access funding provided by Università degli Studi di Verona within the CRUI-CARE Agreement. Work partially supported by SERICS (PE00000014 – CUP H73C2200089001) and iNEST (ECS00000043 – CUP H43C22000540006) funded by PNRR Next Generation EU.

References

1. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf (2008). Accessed: 06/2023
2. Bonnici, V., Arceri, V., Diana, A., Bertini, F., Iotti, E., Levante, A., Bernini, V., Neviani, E., Dal Palù, A.: Biochain: towards a platform for securely sharing microbiological data. In: Proceedings of the 27th International Database Engineered Applications Symposium. IDEAS '23, pp. 59–63. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3589462.3589501
3. Kar, A.K., Navin, L.: Diffusion of blockchain in insurance industry: an analysis through the review of academic and trade literature. Telemat. Inform. **58**, 101532 (2021). https://doi.org/10.1016/j.tele.2020.101532
4. Mühle, A., Grüner, A., Gayvoronskaya, T., Meinel, C.: A survey on essential components of a self-sovereign identity. Comput. Sci. Rev. **30**, 80–86 (2018). https://doi.org/10.1016/j.cosrev.2018.10.002
5. Saberi, S., Kouhizadeh, M., Sarkis, J., Shen, L.: Blockchain technology and its relationships to sustainable supply chain management. Int. J. Prod. Res. **57**(7), 2117–2135 (2019). https://doi.org/10.1080/00207543.2018.1533261
6. Al-Jaroodi, J., Mohamed, N.: Blockchain in industries: a survey. IEEE Access **7**, 36500–36515 (2019). https://doi.org/10.1109/ACCESS.2019.2903554
7. Porru, S., Pinna, A., Marchesi, M., Tonelli, R.: Blockchain-oriented software engineering: challenges and new directions. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 169–171 (2017). https://doi.org/10.1109/icse-c.2017.142
8. Bosu, A., Iqbal, A., Shahriyar, R., Chakraborty, P.: Understanding the motivations, challenges and needs of blockchain software developers: a survey. Empir. Softw. Eng. **24**(4), 2636–2673 (2019). https://doi.org/10.1007/s10664-019-09708-7
9. Popper, N.: A hacking of more than $50 million dashes hopes in the world of virtual currency. The New York Times. June 17th (2016)
10. Destefanis, G., Marchesi, M., Ortu, M., Tonelli, R., Bracciali, A., Hierons, R.: Smart contracts vulnerabilities: a call for blockchain software engineering? In: 2018 International Workshop on Blockchain Oriented Software Engineering (IWBOSE), pp. 19–25 (2018). https://doi.org/10.1109/IWBOSE.2018.8327567
11. Lantz, L., Cawrey, D.: Mastering Blockchain: Unlocking the Power of Cryptocurrencies, Smart Contracts, and Decentralized Applications. O'Reilly (2020)
12. Antonopoulos, A.M.: Mastering Bitcoin: Programming the Open Blockchain, 2nd edn. O'Reilly, Sebastopol (2017)
13. Buterin, V.: Ethereum whitepaper. https://ethereum.org/en/whitepaper/ (2013). Accessed: 06/2023
14. Antonopoulos, A.M., Wood, G.: Mastering Ethereum: Building Smart Contracts and Dapps. O'Reilly, Sebastopol (2018)
15. Goodman, L.M.: Tezos whitepaper (2014). https://tezos.com/whitepaper.pdf
16. Allombert, V., Bourgoin, M., Tesson, J.: Introduction to the Tezos blockchain. In: 2019 International Conference on High Performance Computing and Simulation (HPCS), pp. 1–10 (2019). https://doi.org/10.1109/hpcs48598.2019.9188227
17. Hyperledger: Hyperledger fabric documentation. https://hyperledger-fabric.readthedocs.io/en/release-2.2/blockchain.html#what-is-hyperledger-fabric. Accessed: 10/2022
18. Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., De Caro, A., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolić, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference. EuroSys '18. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3190508.3190538
19. Tendermint: What is tendermint. https://docs.tendermint.com/v0.33/introduction/what-is-tendermint.html (2020). Accessed: 10/2022
20. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. PhD thesis, University of Guelph (2016)
21. Lamport, L., Shostak, R., Pease, M.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982). https://doi.org/10.1145/357172.357176
22. Aggarwal, S., Kumar, N.: Introduction to blockchain. In: The Blockchain Technology for Secure and Smart Applications Across Industry Verticals. Advances in Computers, vol. 121, pp. 211–226. Elsevier, Amsterdam (2021)
23. Marijan, D., Lal, C.: Blockchain verification and validation: techniques, challenges, and research directions. Comput. Sci. Rev. **45**, 100492 (2022). https://doi.org/10.1016/j.cosrev.2022.100492

24. Piscitello, D.M., Chapin, A.L.: Open Systems Networking: TCP/IP and OSI. Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, Reading (1993)
25. Robinson, P.: Survey of crosschain communications protocols. Comput. Netw. **200**, 108488 (2021). https://doi.org/10.1016/j.comnet.2021.108488
26. Hassan, S., De Filippi, P.: Decentralized autonomous organization. Int. Policy Rev. **10**(2), 1–10 (2021). https://doi.org/10.14763/2021.2.1556
27. Min, T., Wang, H., Guo, Y., Cai, W.: Blockchain games: a survey. In: 2019 IEEE Conference on Games (CoG), pp. 1–8 (2019). https://doi.org/10.1109/cig.2019.8848111
28. Min, T., Cai, W.: A security case study for blockchain games. In: 2019 IEEE Games, Entertainment, Media Conference (GEM), pp. 1–8 (2019). https://doi.org/10.1109/gem.2019.8811555
29. Fowler, M., Foemmel, M.: Continuous Integration (2006)
30. Mahdi, H., Miraz, M.A.: Blockchain enabled smart contract based applications: deficiencies with the software development life cycle models. Baltica **33**, 101–116 (2020)
31. Bosu, A., Iqbal, A., Shahriyar, R., Chakraborty, P.: Understanding the motivations, challenges and needs of blockchain software developers: a survey. Empir. Softw. Eng. **24**(4), 2636–2673 (2019). https://doi.org/10.1007/s10664-019-09708-7
32. Olivieri, L., Tagliaferro, F., Arceri, V., Ruaro, M., Negrini, L., Cortesi, A., Ferrara, P., Spoto, F., Talin, E.: Ensuring determinism in blockchain software with GoLiSA: an industrial experience report. In: Proceedings of the 11th ACM SIGPLAN International Workshop on the State of the Art in Program Analysis. SOAP 2022, pp. 23–29. Association for Computing Machinery, New York (2022). https://doi.org/10.1145/3520313.3534658
33. Olivieri, L., Negrini, L., Arceri, V., Tagliaferro, F., Ferrara, P., Cortesi, A., Spoto, F.: Information flow analysis for detecting non-determinism in blockchain. In: Ali, K., Salvaneschi, G. (eds.) 37th European Conference on Object-Oriented Programming (ECOOP 2023). Leibniz International Proceedings in Informatics (LIPIcs), vol. 263, pp. 1–25. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl (2023). https://doi.org/10.4230/LIPIcs.ECOOP.2023.23
34. Interchain Foundation: Cosmos network. https://cosmos.network/ (2024). Accessed 04/2024
35. Liu, J., Liu, Z.: A survey on security verification of blockchain smart contracts. IEEE Access **7**, 77894–77904 (2019). https://doi.org/10.1109/ACCESS.2019.2921624
36. Zhang, R., Xue, R., Liu, L.: Security and privacy on blockchain. ACM Comput. Surv. **52**(3), 1–34 (2019). https://doi.org/10.1145/3316481
37. Guo, H., Yu, X.: A survey on blockchain technology and its security. Blockchain: Res. Appl. **3**(2), 100067 (2022). https://doi.org/10.1016/j.bcra.2022.100067
38. Julien Bertrane, J., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.: Static analysis by abstract interpretation of embedded critical software. SIGSOFT Softw. Eng. Notes **36**(1), 1–8 (2011). https://doi.org/10.1145/1921532.1921553
39. Chakraborty, P., Shahriyar, R., Iqbal, A., Bosu, A.: Understanding the software development practices of blockchain projects: a survey. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ESEM '18. Association for Computing Machinery, New York (2018). https://doi.org/10.1145/3239235.3240298
40. Patrick, C.: Principles of Abstract Interpretation. MIT Press Academic, Cambridge (2021)
41. Rival, X., Yi, K.: Introduction to Static Analysis: An Abstract Interpretation Perspective. Mit Press, Cambridge (2020)
42. Clarke, E.M. Jr., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (1999)
43. Gallier, J.H.: Logic for Computer Science: Foundations of Automatic Theorem Proving. Courier Dover Publications, Mineola (2015)
44. Hähnle, R., Huisman, M.: Deductive software verification: from pen-and-paper proofs to industrial tools. In: Computing and Software Science: State of the Art and Perspectives, pp. 345–373 (2019). https://doi.org/10.1007/978-3-319-91908-9_18
45. Murray, Y., Anisi, D.A.: Survey of formal verification methods for smart contracts on blockchain. In: 2019 10th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–6 (2019). https://doi.org/10.1109/NTMS.2019.8763832
46. Schneidewind, C., Grishchenko, I., Scherer, M., Maffei, M.: eThor: practical and provably sound static analysis of Ethereum smart contracts. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. CCS '20, pp. 621–640. Association for Computing Machinery, New York (2020). https://doi.org/10.1145/3372297.3417250
47. Wesley, S., Christakis, M., Navas, J.A., Trefler, R., Wüstholz, V., Gurfinkel, A.: Verifying solidity smart contracts via communication abstraction in smartACE. In: Finkbeiner, B., Wies, T. (eds.) Verification, Model Checking, and Abstract Interpretation, pp. 425–449. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94583-1_21
48. Otoni, R., Marescotti, M., Alt, L., Eugster, P., Hyvärinen, A., Sharygina, N.: A solicitous approach to smart contract verification. ACM Trans. Priv. Secur. **26**(2), 1–28 (2023). https://doi.org/10.1145/3564699
49. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (1969). https://doi.org/10.1145/363235.363259
50. Barrett, C., Tinelli, C.: Satisfiability modulo theories. In: Handbook of Model Checking, pp. 305–343 (2018)
51. Ferrara, P., Negrini, L., Arceri, V., Cortesi, A.: Static analysis for dummies: experiencing LiSA. In: Proceedings of the 10th ACM SIGPLAN International Workshop on the State of the Art in Program Analysis. Soap 2021, pp. 1–6. Association for Computing Machinery, New York (2021). https://doi.org/10.1145/3460946.3464316
52. Marchesi, L., Marchesi, M., Tonelli, R.: ABCDE – agile block chain DApp engineering. Blockchain: Res. Appl. **1**(1), 100002 (2020). https://doi.org/10.1016/j.bcra.2020.100002
53. Marchesi, L., Marchesi, M., Pompianu, L., Tonelli, R.: Security checklists for Ethereum smart contract development: patterns and best practices (2020). https://doi.org/10.48550/arXiv.2008.04761
54. Baralla, G., Pinna, A., Corrias, G.: Ensure traceability in European food supply chain by using a blockchain system. In: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), pp. 40–47 (2019). https://doi.org/10.1109/WETSEB.2019.00012
55. Mudge, N.: ERC-2535: diamonds, multi-facet proxy. https://eips.ethereum.org/EIPS/eip-2535. Accessed: 06/2023
56. OpenZeppelin: Proxy upgrade pattern. https://docs.openzeppelin.com/upgrades-plugins/1.x/proxies. Accessed: 06/2023
57. Fabric, H.: Upgrade a chaincode. https://hyperledger-fabric.readthedocs.io/en/release-2.5/chaincode_lifecycle.html#upgrade-a-chaincode (2023). Accessed 02/2024
58. Olivieri, L., Spoto, F., Tagliaferro, F.: On-chain smart contract verification over tendermint. In: 5th Wokshop on Trusted Smart Contracts (WTSC'21). Lecture Notes in Computer Science, vol. 12676, pp. 333–347. Springer, Berlin (2021). https://doi.org/10.1007/978-3-662-63958-0_28
59. Arceri, V., Dolcetti, G., Zaffanella, E.: Speeding up static analysis with the split operator. In: Proceedings of the 12th ACM SIGPLAN International Workshop on the State of the Art in Program

Analysis. SOAP 2023, pp. 14–19. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3589250.3596141
60. Atzei, N., Bartoletti, M., Cimoli, T.: A survey of attacks on Ethereum smart contracts (SoK). In: Maffei, M., Ryan, M. (eds.) Principles of Security and Trust, pp. 164–186. Springer, Berlin (2017). https://doi.org/10.1007/978-3-662-54455-6_8
61. Yamashita, K., Nomura, Y., Zhou, E., Pi, B., Jun, S.: Potential risks of hyperledger fabric smart contracts. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), pp. 1–10 (2019). https://doi.org/10.1109/iwbose.2019.8666486
62. Barboni, M., Morichetta, A., Polini, A.: Smart contract testing: challenges and opportunities. In: Proceedings of the 5th International Workshop on Emerging Trends in Software Engineering for Blockchain. WETSEB '22, pp. 21–24. Association for Computing Machinery, New York (2023). https://doi.org/10.1145/3528226.3528370
63. Olivieri, L., Jensen, T., Negrini, L., Spoto, F.: MichelsonLiSA: a static analyzer for Tezos. In: 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops), pp. 80–85 (2023). https://doi.org/10.1109/PerComWorkshops56833.2023.10150247
64. Olivieri, L., Negrini, L., Arceri, V., Jensen, T., Spoto, F.: Design and implementation of static analyses for Tezos smart contracts. Distrib. Ledger Technol. (2024). Just Accepted. https://doi.org/10.1145/3643567
65. MITRE: CVE-2010-5139. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-5139 (2010). Accessed: 06/2023
66. MITRE: CVE-2018-11687. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-11687 (2018). Accessed: 06/2023
67. MITRE: CVE-2018-10299. https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2018-10299 (2018). Accessed: 06/2023
68. Lv, Z., Wu, D., Yang, W., Duan, L.: Attack and protection schemes on fabric isomorphic crosschain systems. Int. J. Distrib. Sens. Netw. **18**(1), 15501477211059945 (2022)
69. Negrini, L., Ferrara, P., Arceri, V., Cortesi, A.: Lisa: a generic framework for multilanguage static analysis. In: Proceedings of 1st Challenges of Software Verification (2023). https://doi.org/10.1007/978-981-19-9601-6_2
70. Buro, S., Crole, R., Mastroeni, I.: On multi-language abstraction: towards a static analysis of multi-language programs. Form. Methods Syst. Des., 1–35 (2023). https://doi.org/10.1007/s10703-022-00405-8
71. Ferrara, P., Negrini, L.: Sarl: OO framework specification for static analysis. In: Software Verification, pp. 3–20. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-63618-0_1
72. Bartoletti, M., Benetollo, L., Bugliesi, M., Crafa, S., Sasso, G.D., Pettinau, R., Pinna, A., Piras, M., Rossi, S., Salis, S., et al.: Smart contract languages: a comparative analysis (2024). arXiv preprint arXiv:2404.04129. https://doi.org/10.48550/arXiv.2404.04129
73. IOTA: EVM smart contracts. https://wiki.iota.org/isc/getting-started/languages-and-vms/#evm-smart-contracts (2024). Accessed 02/2024
74. Alshaikhli, M., Elfouly, T., Elharrouss, O., Mohamed, A., Ottakath, N.: Evolution of Internet of Things from blockchain to IOTA: a survey. IEEE Access **10**, 844–866 (2021). https://doi.org/10.1109/ACCESS.2021.3138353
75. ArcheType. https://archetype-lang.org/ (2024). Accessed 04/2024
76. LIGO: LIGO documentation. https://ligolang.org/ (2024). Accessed 04/2024
77. SmartPy. https://smartpy.io/docs/ (2024). Accessed 04/2024
78. Nomadic Labs: Michelson: the language of smart contracts in Tezos. https://tezos.gitlab.io/active/michelson.html#michelson-the-language-of-smart-contracts-in-tezos (2023). Accessed 04/2023
79. World Wide Web Consortium: WebAssembly overview. https://webassembly.org (2024). Accessed 04/2024
80. CosmWasm: CosmWasm book. https://book.cosmwasm.com/ (2024). Accessed 04/2024
81. Parity Technologies: Ink! documentation. https://paritytech.github.io/ink-docs/why-rust-for-smart-contracts (2024). Accessed 04/2024
82. Web3 Foundation: Polkadot network. https://polkadot.network/ (2024). Accessed 04/2024
83. IOTA: Wasm VM for ISC. https://wiki.iota.org/isc/getting-started/languages-and-vms/#wasm-vm-for-isc (2024). Accessed 02/2024
84. Genet, T., Jensen, T., Sauvage, J.: Termination of Ethereum's smart contracts. In: Proceedings of the 17th International Joint Conference on e-Business and Telecommunications – SECRYPT, pp. 39–51. SciTePress, Setúbal (2020). INSTICC. https://doi.org/10.5220/0009564100390051
85. Wood, G., et al.: Ethereum: a secure decentralised generalised transaction ledger. Ethereum Proj. Yellow Pap. **151**(2014), 1–32 (2014)
86. Kiayias, A., Miller, A., Zindros, D.: Non-interactive proofs of proof-of-work. In: Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020. Revised Selected Papers, vol. 24, pp. 505–522. Springer, Berlin (2020). https://doi.org/10.1007/978-3-030-51280-4_27
87. Maung Maung Thin, W.Y., Dong, N., Bai, G., Dong, J.S.: Formal analysis of a proof-of-stake blockchain. In: 2018 23rd International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 197–200 (2018). https://doi.org/10.1109/ICECCS2018.2018.00031
88. Yoo, J., Jung, Y., Shin, D., Bae, M., Jee, E.: Formal modeling and verification of a federated Byzantine agreement algorithm for blockchain platforms. In: 2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE), pp. 11–21 (2019). https://doi.org/10.1109/IWBOSE.2019.8666514
89. Foundation, S.D.: Intro to stellar. https://stellar.org/learn/intro-to-stellar. Accessed 05/2024
90. Kawahara, R.: Verification of customizable blockchain consensus rule using a formal method. In: 2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1–3 (2020). https://doi.org/10.1109/ICBC48266.2020.9169472
91. European Parliament and the Council: Regulation (EU) 2023/2854 of the European Parliament and of the Council of 13 December 2023 on harmonised rules on fair access to and use of data and amending Regulation (EU) 2017/2394 and Directive (EU) 2020/1828 (Data Act). Document 32023R2854. PE/49/2023/REV/1 OJ L, 2023/2854, 22.12.2023, ELI: http://data.europa.eu/eli/reg/2023/2854/oj (2023)
92. European Parliament and the Council: Consolidated text: Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). Document 02016R0679-20160504. ELI: http://data.europa.eu/eli/reg/2016/679/2016-05-04 (2016)
93. Olivieri, L., Pasetto, L.: Towards compliance of smart contracts with the European Union data act. In: 5th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata,

and Synthesis (OVERLAY 2023). CEUR Workshop Proceedings, vol. 3629, pp. 61–66 (2024). https://ceur-ws.org/Vol-3629/paper10.pdf

94. Haque, A.B., Islam, A.K.M.N., Hyrynsalmi, S., Naqvi, B., Smolander, K.: GDPR compliant blockchains–a systematic literature review. IEEE Access **9**, 50593–50606 (2021). https://doi.org/10.1109/ACCESS.2021.3069877
95. Molina, F., Betarte, G., Luna, C.: Design principles for constructing GDPR-compliant blockchain solutions. In: 2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), pp. 1–8 (2021). https://doi.org/10.1109/WETSEB52558.2021.00008
96. Tauqeer, A., Kurteva, A., Chhetri, T.R., Ahmeti, A., Fensel, A.: Automated GDPR contract compliance verification using knowledge graphs. Information **13**(10), 447 (2022). https://doi.org/10.3390/info13100447
97. Chhetri, T.R., Kurteva, A., DeLong, R.J., Hilscher, R., Korte, K., Fensel, A.: Data protection by design tool for automated GDPR compliance verification based on semantically modeled informed consent. Sensors **22**(7), 2763 (2022). https://doi.org/10.3390/s22072763
98. Ferrara, P., Spoto, F.: Static analysis for GDPR compliance. In: CEUR Workshop Proceedings – Proceedings of ITASEC ’18, vol. 2058, pp. 1–10 (2018). https://ceur-ws.org/Vol-2058/paper-10.pdf
99. Ferrara, P., Olivieri, L., Spoto, F.: Tailoring taint analysis to GDPR. In: Medina, M., Mitrakas, A., Rannenberg, K., Schweighofer, E., Tsouroulas, N. (eds.) Privacy Technologies and Policy, pp. 63–76. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-02547-2_4
100. Ferrara, P., Olivieri, L., Spoto, F.: Static privacy analysis by flow reconstruction of tainted data. Int. J. Softw. Eng. Knowl. Eng. **31**(7), 973–1016 (2021). https://doi.org/10.1142/S0218194021500303
101. Hewa, T., Ylianttila, M., Liyanage, M.: Survey on blockchain based smart contracts: applications, opportunities and challenges. J. Netw. Comput. Appl. **177**, 102857 (2021). https://doi.org/10.1016/j.jnca.2020.102857
102. Zheng, Z., Xie, S., Dai, H.-N., Chen, W., Chen, X., Weng, J., Imran, M.: An overview on smart contracts: challenges, advances and platforms. Future Gener. Comput. Syst. **105**, 475–491 (2020). https://doi.org/10.1016/j.future.2019.12.019
103. Khan, S.N., Loukil, F., Ghedira-Guegan, C., Benkhelifa, E., Bani-Hani, A.: Blockchain smart contracts: applications, challenges, and future trends. Peer-to-Peer Netw. Appl. **14**, 2901–2925 (2021). https://doi.org/10.1007/s12083-021-01127-0
104. Fotiou, N., Polyzos, G.C.: Smart contracts for the Internet of Things: opportunities and challenges. In: 2018 European Conference on Networks and Communications (EuCNC), pp. 256–260 (2018). https://doi.org/10.1109/EuCNC.2018.8443212
105. Zou, W., Lo, D., Kochhar, P.S., Le, X.-B.D., Xia, X., Feng, Y., Chen, Z., Xu, B.: Smart contract development: challenges and opportunities. IEEE Trans. Softw. Eng. **47**(10), 2084–2106 (2021). https://doi.org/10.1109/TSE.2019.2942301
106. Bosu, A., Iqbal, A., Shahriyar, R., Chakraborty, P.: Understanding the motivations, challenges and needs of blockchain software developers: a survey. Empir. Softw. Eng. **24**(4), 2636–2673 (2019). https://doi.org/10.1007/s10664-019-09708-7
107. Singh, A., Parizi, R.M., Zhang, Q., Choo, K.-K.R., Dehghantanha, A.: Blockchain smart contracts formalization: approaches and challenges to address vulnerabilities. Comput. Secur. **88**, 101654 (2020). https://doi.org/10.1016/j.cose.2019.101654
108. Krichen, M., Lahami, M., Al–Haija, Q.A.: Formal methods for the verification of smart contracts: a review. In: 2022 15th International Conference on Security of Information and Networks (SIN), pp. 01–08 (2022). https://doi.org/10.1109/SIN56466.2022.9970534
109. Marijan, D., Lal, C.: Blockchain verification and validation: techniques, challenges, and research directions. Comput. Sci. Rev. **45**, 100492 (2022). https://doi.org/10.1016/j.cosrev.2022.100492
110. Lin, I.-C., Liao, T.-C.: A survey of blockchain security issues and challenges. Int. J. Netw. Secur. **19**(5), 653–659 (2017)
111. Islam, M.R., Rahman, M.M., Mahmud, M., Rahman, M.A., Mohamad, M.H.S., Embong, A.H.: A review on blockchain security issues and challenges. In: 2021 IEEE 12th Control and System Graduate Research Colloquium (ICSGRC), pp. 227–232 (2021). https://doi.org/10.1109/ICSGRC53186.2021.9515276
112. Magazzeni, D., McBurney, P., Nash, W.: Validation and verification of smart contracts: a research agenda. Computer **50**(9), 50–57 (2017). https://doi.org/10.1109/MC.2017.3571045
113. Koul, R.: Blockchain oriented software testing – challenges and approaches. In: 2018 3rd International Conference for Convergence in Technology (I2CT), pp. 1–6 (2018). https://doi.org/10.1109/I2CT.2018.8529728