Cross-chain Smart Contracts and dApps Verification by Static Analysis: Limits and Challenges

Luca Olivieri[1], Aradhita Mukherjee[1], Nabendu Chaki[2] and Agostino Cortesi[1]

[1]*Ca' Foscari University of Venice, Venice, Italy*
[2]*University of Calcutta, Kolkata, India*

Abstract

The increasing demand for interoperability, security, and reliability among various blockchain ecosystems has necessitated the development of cross-chain verification techniques. Static analysis, a powerful tool in software development, offers a promising approach to identify bugs, vulnerabilities, and inconsistencies in cross-chain software during its early stages. By detecting these issues proactively, static analysis can significantly reduce development costs, time-to-market, and the risk of deploying immutable, buggy smart contracts. However, providing adequate verification for cross-chain software is not straightforward. It involves analyzing diverse components, understanding complex program behaviors, interpreting different instruction semantics, and navigating the intricacies of various programming languages. This paper aims to provide a comprehensive overview of interoperability in the blockchain context. We delve into the state-of-the-art regarding verification tools specifically designed for cross-chain software and explore the known issues that these tools can effectively detect. Furthermore, we highlight potential pitfalls and challenges associated with static analysis in this domain, offering insights into the complexities and limitations of this approach.

Keywords

Blockchain, distributed ledger technology, smart contracts, decentralized applications, multi-chain, multi-blockchain, cross-chain, cross-blockchain, blockchain interoperability, cross-program verification, program verification, static analysis, software security, software verification, vulnerability detection.

1. Introduction

The rapid digital expansion of recent years has led to a surge in cyber challenges and threats worldwide [1], highlighting the importance of blockchain technology in ensuring data integrity, security and trust in digital transactions.

A blockchain is a decentralized and distributed digital ledger shared among a network that securely records transactions across multiple peers. It operates without a central authority and ensures transparency, anti-tampering, and data immutability. Smart contracts are computer programs that can be deployed, typically in an immutable way, and executed within the blockchain. Decentralized applications (dApps) are applications that run on blockchain and leverage smart contracts to provide decentralized, transparent, and trustless services.

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT
luca.olivieri@unive.it (L. Olivieri); aradhita.mukherjee@unive.it (A. Mukherjee); nabendu@ieee.org (N. Chaki); cortesi@unive.it (A. Cortesi)
0000-0001-8074-8980 (L. Olivieri); 0000-0001-8787-9086 (A. Mukherjee); 0000-0003-3242-680X (N. Chaki); 0000-0002-0946-5440 (A. Cortesi)

Achieving cross-blockchain interoperability is a non-trivial task due to the heterogeneous diffusion of blockchain solutions and the lack of regulatory standards and unified communication protocols [2]. For these reasons, the development of interoperable solutions is error-prone and can introduce bugs and vulnerabilities in the code. Just consider that, as of mid-2024, more than $2.8 billion has been lost due to cross-chain exploits [3, 4, 5]. Software verification is crucial in these settings, but it is also challenging, and the existing tools lack static analyses that can fully account for the complexities of the cross-chain paradigm [6].

Static analysis allows one to verify software without executing its code, i.e., statically [7]. Typically, verification tools based on static analysis techniques issue alerts to highlight program points of interest, such as instructions with potential errors, bugs, and vulnerabilities. In software development, static analysis may significantly reduce costs by detecting issues in an early stage of coding without the overhead of setting up full-scale test environments.

The blockchain operates in a *trustless environment* where blockchain peers and users do not necessarily trust each other. Hence, most blockchains allow the deployment of smart contracts only in an immutable way. Consequently, code fixing and patching may not always be feasible or might be difficult to apply. Thus, static analysis takes a crucial role in the blockchain context. Static analysis helps avoid the immutable deployment of buggy smart contracts by catching issues early. This also minimizes the risk of exploiting vulnerabilities and critical failures of live dApps in blockchain systems. This, in turn, helps to reduce the cost of re-deployment in case of fee-based blockchains.

In this paper, we investigate the current state of verification in cross-chain software. We highlight existing approaches, their limitations, and potential areas for improvement related to static analysis. Furthermore, it briefly overviews cross-chain properties (e.g. bugs, vulnerabilities, and inconsistencies) that can be detected with static analysis.

Paper Structure The rest of the paper is structured as follows. Section 2 offers an overview of cross-chain environments and the different kinds of blockchain interoperability. Section 3 discusses the state-of-art and limitations of the existing verification tools for cross-chain solutions. Section 4 deals with a sound representation of code during analysis. Section 5 investigates interesting properties for cross-chain verification. Finally, Section 6 concludes the paper.

2. Blockchain Interoperability

Blockchain interoperability allows different blockchain networks to communicate, share data, and exchange assets with each other. However, achieving seamless interoperability poses several challenges. In fact, not all blockchains were initially designed for interaction with other ecosystems. Early blockchains like Bitcoin [8, 9] and Ethereum [10, 11] were built as isolated, self-contained networks with no inherent ability to communicate with other blockchains. Then, they usually require external solutions, like *bridges* [4], *oracles* [12] and *relays* [13] to achieve interoperability. On the other hand, more recent blockchains such as Cosmos [14] and Polkadot [15] already provide *native interoperability protocols* [16], where blockchains can communicate easily within their own multi-chain ecosystems. However, the latter have not reached the same diffusion level as the former, having emerged only later. In fact, currently, the

Table 1
Cross-Chain Verification Tools

Tool	Kind of Verification	Kind of Interoperability	External Solutions	Languages	IR	Merging Points	Sound Construction
XGuard [21]	Static Analysis	Homogeneus (Ethereum-based only)	Code not included in the analysis	Solidity	CFG	Event-driven	✗
SmartAxe [22]	Static Analysis	Homogeneus (Ethereum-based only)	Code not included in the analysis	EVM bytecode	CFG	Event-driven	✗
ChainSniper [23]	Hybrid (Machine Learning)	Homogeneus (Ethereum-based only)	Code not included in the analysis	Solidity	CFG (for ML models)	n.d.	✗

blockchain economy still revolves around blockchains based on the Ethereum protocol, which hold the highest levels of Total Value Locked (TVL) in terms of decentralized finance (DeFi) such as financial assets and cryptocurrencies [17].

In blockchain, interoperability can be divided into two categories: *homogeneous* and *heterogeneous*. The first case means interoperability between blockchains with similar or identical architectures. These blockchains typically share common protocols, consensus mechanisms, or environments, making communicating or sharing data and assets easier. For instance, homogeneous interoperability solutions connect Ethereum-based blockchains such as Ethereum (main-net), Tron, and Polygon. Instead, heterogeneous interoperability refers to the ability of blockchains with different architectures, consensus mechanisms, and virtual machines to interact. These blockchains were often built with distinct goals and structures, making cross-chain communication more complex, suitable, and flexible. For instance, heterogeneous interoperability solutions are those that connect Ethereum with Bitcoin, Cosmos, etc.

3. The State-of-Art in Cross-Chain Verification

In recent years, the verification of smart contracts has matured significantly, evolving from rudimentary techniques involving only syntactic checks to sophisticated methodologies that enhance security and reliability by modelling program behaviours and instruction semantics.

Although smart contract verification has moved beyond its infancy, currently, the tools are mainly focused on individual and mainstream blockchains, making a gap on the less “popular” ones [18]. According to [19, 6], developing semantic-based tools requires a significant theoretical background and programming skills, even to provide toy implementations, making it difficult for traditional developers and blockchain practitioners. This leads to only a few blockchains benefiting from adequate verification tools, and the others are not supported in the same way. Furthermore, interoperability is still not considered a strong requirement in the development of verification tools.

In general, verifying individual components prevents local issues and domain-specific vulnerabilities, but there are issues which could not be discovered by analyzing each program in isolation only [20]. Indeed, this is not sufficient also in cross-chain settings where contracts interact with other blockchains and software components outside the blockchain where they are deployed.

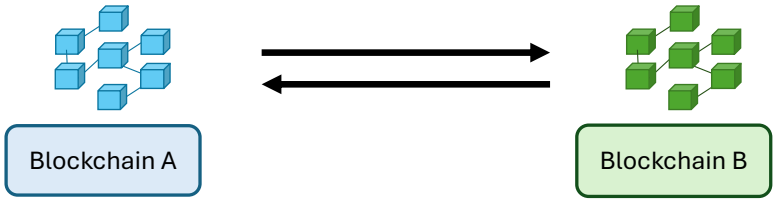
To the best of our knowledge, only a few recent studies deal with verifying interoperable blockchain solutions. Focusing on static analysis, as reported in Table 1, state-of-the-art is currently limited to homogenous interoperability between Ethereum-based blockchains, and they consider only events from bridge solutions as cross-chain connectors. No heterogeneous

interoperability and other external solutions are currently covered. For instance, XGuard [21] extracts and analyzes the semantic information and the source/destination blockchains of bridging operations from Ethereum-based bridge contracts written in Solidity, a high-level language for the development of Ethereum smart contracts. To do that, it exploits under the woods the static analyzer Slither [24] and then provides additional verification modules to detect inconsistent behaviours. Also, SmartAxe [22] identifies vulnerabilities in cross-chain bridge smart contracts but written EVM bytecode, a low-level language of Ethereum smart contracts. It performs control-flow and data-flow analyses to detect issues such as access control incompleteness and semantic inconsistencies. However, the main drawback of these tools do not provide guarantees about findings because they suffer from both false positives (i.e. spurious, not real results) and false negatives (i.e. real results not detected). The reason is that they lack to properly apply formal methods that allow one to achieve *soundness* (i.e. absence of false negatives) or *completeness* (i.e. absence of false positives) [7].

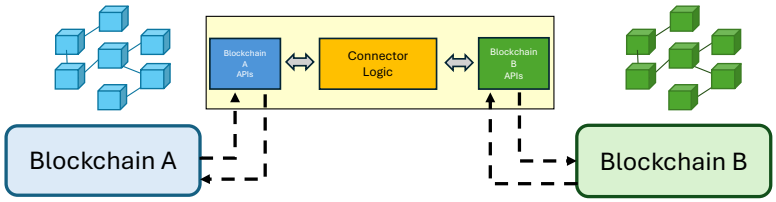
Static analysis can also be adopted to support artificial intelligence (AI) approaches. In particular, according to Ressi et al. [25, 26], static analysis is widely involved in the training of machine-learning (ML) models to build graph-based representations of the code, such as the control-flow graphs (CFGs) [27], to feed neural networks or combined with graph neural network models. Additionally, static analysis tools can also be applied to label train datasets for deep learning models. For instance, ChainSniper [23] enables the secure exchange of data between interoperable blockchains through automated AI-based analysis, and its ML model is trained using static analysis as well. However, compared with traditional static analysis, there are no strong guarantees about the findings as verification through ML relies on probabilistic models rather than rigorous formal techniques.

4. Towards an Adequate Representation

The main step for any effective static analysis is the extraction of an intermediate representation (IR) from the software to analyse that provides an abstracted form of code that is easier for analysis tools to work with and typically represents complex programming constructs, structures, and semantics in a way that is agnostic from the programming language of the software to analyze. Then, it can be analyzed to infer code properties and to detect bugs, issues, and vulnerabilities. For instance, the CFG is one of the most used IR in blockchain software verification [28]. It is a directed graph that is exploited to understand how the program executes and is used to verify that the source code is syntactically and semantically correct. CFG models the control flow between different blocks or statements in the code, modelling the various branches, loops, and jumps that can occur during execution. Each path in the CFG corresponds to a possible program execution. To perform adequate cross-chain verification, it is necessary to know the entire execution flow of the cross-chain interaction. It is not sufficient to compute the individual IR for each party (e.g., smart contracts, external components, ...) involved in the cross-chain interaction; it is necessary to merge them properly. Merging multiple IRs may seem like a simple operation. For instance, for CFGs, it might be sufficient to use labelled edges to connect the various nodes of interest and then join the CFGs, being graph representations. However, this is possible under certain assumptions, i.e. that (i) the IR is accurate and comprehensive with



(a) without considering external solutions



(b) considering the software of external solutions

Figure 1: Different system representations

respect to what one want to analyze, (ii) the IRs are both compatible with the chosen analysis engine, that (iii) the merge points are known, and that (iv) they are soundly computed.

4.1. Ensuring an Accurate and Aimed Representation

A correct representation of the cross-chain ecosystem is crucial for conducting an effective and reliable analysis. However, cross-chain ecosystems rely on various mechanisms which can increase the complexity of analysis. Omitting non-critical or irrelevant code can improve the performance and efficiency of static analysis by reducing processing time and resource consumption. It can help focus on specific components or high-risk code areas, making the analysis more targeted and manageable. Furthermore, ignoring certain generated or third-party code can prevent unnecessary noise and false positive alerts in the analysis results. However, on the other hand, omitting code may lead to incomplete analysis, potentially missing critical issues or errors. Dependencies and interactions between omitted and analyzed code might be overlooked, leading to incorrect assumptions about the behaviors of the overall system. Then, it is of primary importance to make correct assumptions before to represent the system to analyze.

A typical example of this is whether or not to consider the code of external solutions in the analyses (see Figure 1). For instance, in a token transfer scenario, different blockchains often use distinct data structures and formats. According to [29, 30], token migration can be complex, even for relatively simple code, particularly when different programming languages are involved, as seen in heterogeneous interoperability. Ensuring a correct transfer and migration is crucial. However, data conversion and token validation are typically managed by external solutions, like

bridges. Ideally, this code should also be analyzed, but in many cases, it is provided by trusted companies or services that do not necessarily make it publicly available. Hence, what happens (see Table 1) is to omit them from the analysis under the assumption that cross-chain exchanged data will be properly handled. Therefore considering only the code within the blockchains.

4.2. Analysis Engines and Types of Interoperability

Regarding homogeneous interoperability, blockchains typically have the same instruction set, instruction semantics, data types, and code that runs on them, which is also written in the same target blockchain language. Analyzing this type of system is less difficult, as single existing verification tools can be adapted to support the analysis of the entire software and consider the cross-chain bindings and constraints.

This happens differently if the interoperability is homogeneous but applied by external solutions (e.g. bridges, relayers, ...) or if it is heterogeneous interoperability. In these cases, the analysis engine is typically required to support and handle different components, behaviours, and programming languages. Indeed, using single tools and merging the tool results is imprecise and could lead to unsound approaches as well as be potentially unfeasible due to compatibility issues and technical limitations of the analysis engines. Furthermore, according to [6, 31], this is challenging because there is a lack of techniques that work with different languages and components at the same time. To the best of our knowledge, only a few verification frameworks model this type of problems [32, 33, 31, 34], and even fewer provide support for blockchain features. In particular, LiSA [31, 35, 19] (Library for Static Analysis) supports some programming languages and frameworks for blockchain, such as Ethereum [36], Hyperledger Fabric [37, 38], Tendermint (rebranded Ignite), Cosmos [38, 18], and Tezos [39, 40]. Also, MOPSA [41] (Modular Open Platform for Static Analysis) supports several programming languages but currently the blockchain support is limited to Tezos only [42]. However, currently, both do not provide any implementation for cross-chain analysis. Compiler frameworks, such as LLVM [43], can also be interesting in this sense. Indeed, they compile different programming languages in a single IR that is typically widely supported by different verification tools. However, as far as we know, there is little support for blockchain programming languages and no support for cross-chain features.

4.3. Different Merging Points

Interactions to ensure interoperability between different blockchains can occur in different ways [44, 45, 2], and in some cases, they are not visible within the code of smart contracts. In the simplest cases, such as for homogeneous multi-chain environments, like in Cosmos or Polkadot, interactions occur through the invocation of specific functions or instructions supported by the blockchain framework during code execution. Instead, for individual blockchains, such as Ethereum, there are no specific code instructions or interoperability standards, and thus, developers need to use workarounds to ensure cross-chain communication. For instance, in Ethereum-like blockchains, a widely used stratagem is *events*, i.e., developers can implement logging functionalities allowing external applications to listen to and react to state changes within the blockchain. In particular, an event is identified by the suffix `event` in Solidity [46],

the main programming language for Ethereum smart contracts. An event supports up to three custom parameters that can store data and is triggerable by the instruction `emit`. Then, bridges, oracles, and relays can detect these changes and automatize the interaction between the different blockchains [47] through for instance web APIs [48]. In other blockchains, such as Hyperledger Fabric [49], the implementation of cross-chain interactions is totally at the discretion of the developer as smart contracts can be written using general-purpose languages and their APIs without any limitations whatsoever (although other types of problems may be introduced [50]). Finally, there are also blockchains, like Tezos [51], that do not have a direct event system or other custom ways to ensure cross-chain interaction at the smart contract level. Developers are required to use monitoring tools to detect storage changes, callback contracts, off-chain indexers, and transaction metadata to achieve similar functionality in these cases. Then, there is no explicit trace of this in the code of smart contracts.

From a program analysis perspective, this may require instrumentations, additional components, and supporting analytics to correctly extract, compute, and model the merging points. For instance, the functions called between one contract and another of a cross-chain interaction, parameters, addresses, and the information exchanged are typically values that can be hard-coded or declared within the code. However, these values can be manipulated and modified by instructions and program behaviours depending on the code execution, such as by mathematical (e.g. addition, subtraction, …) and string (e.g. concatenation, substring, …) operations. Then, supporting analyses on different value domains (e.g. for numbers [52] and strings [53]) may required to infer an approximation of the concrete values.

Another aspect to consider is the automation of merging point detection. Given the growing number of new blockchain frameworks and interoperable paradigms, applying standard models to statically analyze their code could be potentially unsound and imprecise. However, modifying or extending the analysis engine for every existing framework to provide automatic checks for merging points is not sustainable because it may require heavy changes to the analysis engine. In traditional verification, this problem is similar to that of third-party library support, which is generally solved by applying specification frameworks [54]. The specification framework is typically agnostic from the analysis engine. Given a specification, the framework automatically generates annotations on code components which can then be interpreted by the analyzer. In this way, developers and blockchain communities can create new specifications to model new frameworks containing both syntactic and semantic rules for annotating the merging points to consider during the analysis.

4.4. Sound Construction and Representation Accuracy

The computation of an accurate IR is a key point of every static analysis because when it is partial or inaccurate, the analysis results will also be. Indeed, in the worst cases, the analysis might miss properties existing in the real code, leaving potential bugs, issues, and vulnerabilities undetected.

Analyze all possible execution paths and ensure the absence of false negatives means pursuing the *soundness* [7]. However, the sound construction of an IR is not a given in the current existing tools [55]. It poses significant challenges due to potentially statically unknown values in the code, which require approximations resulting from the instruction semantics and that affect

different code executions. For instance, Arceri et al. [36] investigate a sound CFG construction for smart contracts written in Ethereum bytecode and highlight the issues and challenges to achieve soundness due to the approximations of unknown jump destinations of some jump instructions in the code.

In the interoperability settings, it is also necessary to ensure that merging points are computed soundly by approximating potential points in case of lack of information or unknown values. Soundness is typically achieved following a conservative approach where over-approximations on the computations are applied. Although over-approximations may have different levels of precision, this approach can introduce spurious results, which lead to false positives, i.e. behaviours that seem possible according to the analysis but which, in reality, cannot occur in any concrete execution of the software. However, in critical contexts such as the blockchain one, it is advisable to have false positives in static analysis rather than to miss the detection of vulnerabilities that, once deployed, may become immutable and pose significant long-term security risks.

5. Target Properties for Cross-chain Analysis

Once an adequate representation is achieved, specific analyses can be tailored to detect cross-chain bugs, vulnerabilities, and inconsistencies. A few examples of properties that can be analyzed statically in cross-chain contexts are reported below:

- *Reachability property*. It refers to the ability to determine whether a particular state or condition in a program can be reached during its execution. This property is crucial in verifying the correctness and security of smart contracts, as well as for cross-chain software. It helps understand if a code execution from a blockchain can lead to the code execution in another blockchain and detect potential issues like unauthorized access or unintended states.
- *Data and Numerical properties*. The exchange of data is critical in cross-chain blockchain ecosystems because it may involve different blockchains, smart contracts, and programming languages. Hence, this may lead to different data types that require conversions or modifications to be digested during the cross-chain interactions. In particular, numerical values are widely adopted for balances, token quantities, transaction counts, and other numerical attributes. Hence, numerical values may lead to critical issues with also economic impacts, such as truncation errors due to type conversions or numerical overflows/underflow [56] that can lead to the loss of crypto-currencies.
- *Time and synchronization properties*. They are interesting for cross-chain token transfers. For instance, time lock mechanisms are widely involved to ensure that both parties complete their part of the exchange within a set period. Contrary to what happens for individual smart contracts or cross-contract operations within the same blockchain, interactions in cross-chains are not possible using the traditional methods to get the timestamp or block height of the blockchain because they are local with respect to the blockchain where the code is executed, and they may give different results depending to the blockchain. This may lead to unexpected behaviours and problems. For instance,

in the case of unsynchronized time windows, the payment for a token whose trading has already expired may occur, which means spending the money without receiving the token. Then, it is necessary to ensure that the code adopts a trusted oracle that yields the same value to all the blockchains involved in the cross-chain transfer. The oracle is typically provided by the external component acting as the connector for the cross-chain interaction.

- *Confidentiality and privacy properties.* They may investigate to detect data leakages. For instance, developers could be interested in detecting the unintentional sharing of sensitive data in a cross-chain iteration between private and public blockchains. This can be a classic use case for information flow analysis [57, 58].

6. Conclusion

The verification of cross-chain solutions is still in an early stage, presenting numerous challenges and opportunities for improvement. Static analysis can be applied to detect issues and inconsistencies during the software development process, offering significant advantages by reducing the time and economic costs associated with identifying and fixing bugs. For instance, static analysis tools can detect vulnerabilities in smart contracts or misalignments in cross-chain software before they are deployed and before code immutability. However, an effective verification requires a comprehensive approach that considers the entire ecosystem, including homogeneous and heterogeneous interoperabilities among different blockchains. It is insufficient to focus solely on individual components or smart contracts with similar characteristics, as the interactions and connections between distinct blockchains are critical to ensuring system and dApps reliability.

To achieve this, the intermediate representations must be computed adequately and soundly, encompassing all potential execution paths, program behaviors, and instruction semantics of the cross-chain software. This allows the analysis to accurately reflects the complexity of cross-chain environments, where minor discrepancies can lead to significant issues in real-world scenarios.

Furthermore, in the evolving cross-chain landscape, there is an urgent need for the establishment of standards and the development of libraries for cross-chain interactions. Such advancements would enable verification tools to identify merging points and interoperability junctions in an automated way. By facilitating the analysis process, these tools could significantly reduce human effort and minimize the risk of oversight, which is particularly valuable given the complexity of cross-chain systems.

In addition to static analysis, formal methods may be employed during the detection of cross-chain properties to provide stronger guarantees such as the soundness.

Acknowledgments

Work partially supported by SERICS (PE00000014 - CUP H73C2200089001) and iNEST (ECS00000043 – CUP H43C22000540006) projects funded by PNRR NextGeneration EU.

Declaration on Generative AI

During the preparation of this work, the authors used AI-based tools (*Grammarly*, *ChatGPT*) in order to: Grammar and spelling check, Paraphrase and reword. After using this tool/service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] M. R. Bozzetti, L. Olivieri, F. Spoto, Cybersecurity impacts of the Covid-19 pandemic in Italy, in: CEUR Workshop Proceedings, volume 2940, 2021, p. 145 – 155. URL: https://ceur-ws.org/Vol-2940/paper13.pdf, proceedings of the Italian Conference on Cybersecurity (ITASEC 2021).

[2] R. Belchior, A. Vasconcelos, S. Guerreiro, M. Correia, A survey on blockchain interoperability: Past, present, and future trends, ACM Comput. Surv. 54 (2021). URL: https://doi.org/10.1145/3471140. doi:10.1145/3471140.

[3] DefiLlama, Total Value Hacked in Bridges, 2024. https://defillama.com/hacks Accessed 06/2024.

[4] L. Olivieri, A. Mukherjee, N. Chaki, A. Cortesi, Blockchain Interoperability through Bridges: A Token Transfer Perspective, in: Proceedings of the 6th International Conference on Blockchain Computing and Applications (BCCA 2024), 2024.

[5] C. G. Harris, Cross-chain technologies: Challenges and opportunties for blockchain interoperability, in: 2023 IEEE International Conference on Omni-layer Intelligent Systems (COINS), 2023, pp. 1–6. doi:10.1109/COINS57856.2023.10189298.

[6] L. Olivieri, F. Spoto, Software verification challenges in the blockchain ecosystem, International Journal on Software Tools for Technology Transfer 26 (2024) 431–444. URL: https://doi.org/10.1007/s10009-024-00758-x. doi:10.1007/s10009-024-00758-x.

[7] X. Rival, K. Yi, Introduction to static analysis: an abstract interpretation perspective, Mit Press, Cambridge, MA, USA, 2020.

[8] S. Nakamoto, Bitcoin: A Peer-to-Peer Electronic Cash System, 2008. https://bitcoin.org/bitcoin.pdf Accessed: 09/2024.

[9] A. M. Antonopoulos, Mastering Bitcoin: Programming the Open Blockchain, 2nd ed., O'Reilly, Sebastopol, CA, USA, 2017.

[10] G. Wood, et al., Ethereum: A secure decentralised generalised transaction ledger, Ethereum project yellow paper 151 (2014) 1–32.

[11] A. M. Antonopoulos, G. Wood, Mastering Ethereum: Building Smart Contracts and Dapps, O'Reilly, Sebastopol, CA, USA, 2018.

[12] S. K. Ezzat, Y. N. M. Saleh, A. A. Abdel-Hamid, Blockchain oracles: State-of-the-art and research directions, IEEE Access 10 (2022) 67551–67572. doi:10.1109/ACCESS.2022.3184726, https://doi.org/10.1109/ACCESS.2022.3184726.

[13] K. Ren, N.-M. Ho, D. Loghin, T.-T. Nguyen, B. C. Ooi, Q.-T. Ta, F. Zhu, Interoperability in blockchain: A survey, IEEE Transactions on Knowledge and Data Engineering 35 (2023) 12750–12769. doi:10.1109/TKDE.2023.3275220.

[14] Interchain Foundation., Cosmos network, 2024. https://cosmos.network/ Accessed 09/2024.
[15] Web3 Foundation, Polkadot network, 2024. https://polkadot.network/ Accessed 09/2024.
[16] I. A. Qasse, M. Abu Talib, Q. Nasir, Inter blockchain communication: A survey, in: Proceedings of the ArabWIC 6th Annual International Conference Research Track, ArabWIC 2019, Association for Computing Machinery, New York, NY, USA, 2019. URL: https://doi.org/10.1145/3333165.3333167. doi:10.1145/3333165.3333167.
[17] DefiLlama, Total value locked all chains, 2024. https://defillama.com/chains Accessed 10/2024.
[18] L. Olivieri, F. Tagliaferro, V. Arceri, M. Ruaro, L. Negrini, A. Cortesi, P. Ferrara, F. Spoto, E. Talin, Ensuring determinism in blockchain software with golisa: an industrial experience report, SOAP 2022, Association for Computing Machinery, New York, NY, USA, 2022, p. 23–29. doi:10.1145/3520313.3534658, https://doi.org/10.1145/3520313.3534658.
[19] P. Ferrara, L. Negrini, V. Arceri, A. Cortesi, 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, Association for Computing Machinery, New York, NY, USA, 2021, p. 1–6. doi:10.1145/3460946.3464316, https://doi.org/10.1145/3460946.3464316.
[20] A. Mandal, P. Ferrara, Y. Khlyebnikov, A. Cortesi, F. Spoto, Cross-program taint analysis for iot systems, in: Proceedings of the 35th Annual ACM Symposium on Applied Computing, SAC '20, Association for Computing Machinery, New York, NY, USA, 2020, p. 1944–1952. URL: https://doi.org/10.1145/3341105.3373924. doi:10.1145/3341105.3373924.
[21] K. Wang, Y. Li, C. Wang, J. Gao, Z. Guan, Z. Chen, Xguard: Detecting inconsistency behaviors of crosschain bridges, in: Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, FSE 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 612–616. URL: https://doi.org/10.1145/3663529.3663809. doi:10.1145/3663529.3663809.
[22] Z. Liao, Y. Nan, H. Liang, S. Hao, J. Zhai, J. Wu, Z. Zheng, Smartaxe: Detecting cross-chain vulnerabilities in bridge smart contracts via fine-grained static analysis, Proc. ACM Softw. Eng. 1 (2024). URL: https://doi.org/10.1145/3643738. doi:10.1145/3643738.
[23] T.-D. Tran, K. A. Vo, D. T. Phan, C. N. Tan, V.-H. Pham, Chainsniper: A machine learning approach for auditing cross-chain smart contracts, in: Proceedings of the 2024 9th International Conference on Intelligent Information Technology, ICIIT '24, Association for Computing Machinery, New York, NY, USA, 2024, p. 223–230. URL: https://doi.org/10.1145/3654522.3654577. doi:10.1145/3654522.3654577.
[24] J. Feist, G. Grieco, A. Groce, Slither: A static analysis framework for smart contracts, in: 2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB), 2019, pp. 8–15. doi:10.1109/WETSEB.2019.00008.
[25] D. Ressi, A. Spanò, L. Benetollo, C. Piazza, M. Bugliesi, S. Rossi, Vulnerability detection in ethereum smart contracts via machine learning: A qualitative analysis, arXiv preprint arXiv:2407.18639 (2024).
[26] D. Ressi, R. Romanello, C. Piazza, S. Rossi, Ai-enhanced blockchain technology: A review of advancements and opportunities, Journal of Network and Computer Applications 225 (2024) 103858. URL: https://www.sciencedirect.com/science/article/pii/S1084804524000353. doi:https://doi.org/10.1016/j.jnca.2024.103858.

[27] F. E. Allen, Control flow analysis, in: Proceedings of a Symposium on Compiler Optimization, Association for Computing Machinery, New York, NY, USA, 1970, p. 1–19. URL: https://doi.org/10.1145/800028.808479. doi:10.1145/800028.808479.

[28] S. S. Kushwaha, S. Joshi, D. Singh, M. Kaur, H.-N. Lee, Ethereum smart contract analysis tools: A systematic review, IEEE Access 10 (2022) 57037–57062. doi:10.1109/ACCESS.2022.3169902.

[29] M. Crosara, L. Olivieri, F. Spoto, F. Tagliaferro, Fungible and non-fungible tokens with snapshots in java, Cluster Computing 26 (2023) 2701–2718. URL: https://doi.org/10.1007/s10586-022-03756-3. doi:10.1007/s10586-022-03756-3.

[30] M. Crosara, L. Olivieri, F. Spoto, F. Tagliaferro, Re-engineering erc-20 smart contracts with efficient snapshots for the java virtual machine, in: 2021 Third International Conference on Blockchain Computing and Applications (BCCA), 2021, pp. 187–194. URL: https://doi.org/10.1109/BCCA53669.2021.9657047. doi:10.1109/BCCA53669.2021.9657047.

[31] L. Negrini, P. Ferrara, V. Arceri, A. Cortesi, LiSA: A Generic Framework for Multilanguage Static Analysis, Springer Nature Singapore, Singapore, 2023, pp. 19–42. doi:10.1007/978-981-19-9601-6_2, https://doi.org/10.1007/978-981-19-9601-6_2.

[32] S. Buro, R. L. Crole, I. Mastroeni, On multi-language abstraction: Towards a static analysis of multi-language programs, in: Static Analysis: 27th International Symposium, SAS 2020, Virtual Event, November 18–20, 2020, Proceedings, Springer-Verlag, Berlin, Heidelberg, 2020, p. 310–332. URL: https://doi.org/10.1007/978-3-030-65474-0_14. doi:10.1007/978-3-030-65474-0_14.

[33] G. Teixeira, J. a. Bispo, F. F. Correia, Multi-language static code analysis on the lara framework, in: Proceedings of the 10th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, SOAP 2021, Association for Computing Machinery, New York, NY, USA, 2021, p. 31–36. doi:10.1145/3460946.3464317, https://doi.org/10.1145/3460946.3464317.

[34] R. Monat, A. Ouadjaout, A. Miné, A multilanguage static analysis of python programs with native c extensions, in: C. Drăgoi, S. Mukherjee, K. Namjoshi (Eds.), Static Analysis, Springer International Publishing, Cham, 2021, pp. 323–345.

[35] L. Negrini, V. Arceri, L. Olivieri, A. Cortesi, P. Ferrara, Teaching through practice: Advanced static analysis with lisa, in: E. Sekerinski, L. Ribeiro (Eds.), Formal Methods Teaching, Springer Nature Switzerland, Cham, 2024, pp. 43–57. doi:10.1007/978-3-031-71379-8_3, https://doi.org/10.1007/978-3-031-71379-8_3.

[36] V. Arceri, S. M. Merenda, G. Dolcetti, L. Negrini, L. Olivieri, E. Zaffanella, Towards a sound construction of evm bytecode control-flow graphs, in: Proceedings of the 26th ACM International Workshop on Formal Techniques for Java-like Programs, FTfJP 2024, Association for Computing Machinery, New York, NY, USA, 2024, p. 11–16. doi:10.1145/3678721.3686227, https://doi.org/10.1145/3678721.3686227.

[37] L. Olivieri, L. Negrini, V. Arceri, B. Chachar, P. Ferrara, A. Cortesi, Detection of phantom reads in hyperledger fabric, IEEE Access 12 (2024) 80687–80697. doi:10.1109/ACCESS.2024.3410019, https://doi.org/10.1109/ACCESS.2024.3410019.

[38] L. Olivieri, L. Negrini, V. Arceri, F. Tagliaferro, P. Ferrara, A. Cortesi, F. Spoto, Information Flow Analysis for Detecting Non-Determinism in Blockchain, in: K. Ali, G. Salvaneschi (Eds.), 37th European Conference on Object-Oriented Programming

(ECOOP 2023), volume 263 of *Leibniz International Proceedings in Informatics (LIPIcs)*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 2023, pp. 1–25. doi:10.4230/LIPIcs.ECOOP.2023.23, https://doi.org/10.4230/LIPIcs.ECOOP.2023.23.

[39] L. Olivieri, T. Jensen, L. Negrini, F. Spoto, Michelsonlisa: A static analyzer for tezos, in: 2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops), 2023, pp. 80–85. doi:10.1109/PerComWorkshops56833.2023.10150247, https://doi.org/10.1109/PerComWorkshops56833.2023.10150247.

[40] L. Olivieri, L. Negrini, V. Arceri, T. Jensen, F. Spoto, Design and implementation of static analyses for tezos smart contracts, Distrib. Ledger Technol. (2024). doi:10.1145/3643567, https://doi.org/10.1145/3643567.

[41] M. Journault, A. Miné, R. Monat, A. Ouadjaout, Combinations of reusable abstract domains for a multilingual static analyzer, in: S. Chakraborty, J. A. Navas (Eds.), Verified Software. Theories, Tools, and Experiments, Springer International Publishing, Cham, 2020, pp. 1–18.

[42] G. Bau, A. Miné, V. Botbol, M. Bouaziz, Abstract interpretation of michelson smart-contracts, in: Proceedings of the 11th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis, SOAP 2022, Association for Computing Machinery, New York, NY, USA, 2022, p. 36–43. doi:10.1145/3520313.3534660, https://doi.org/10.1145/3520313.3534660.

[43] L. Foundation, The LLVM Compiler Infrastructure, 2024. https://llvm.org/ Accessed 09/2024.

[44] H. Mao, T. Nie, H. Sun, D. Shen, G. Yu, A survey on cross-chain technology: Challenges, development, and prospect, IEEE Access 11 (2023) 45527–45546. doi:10.1109/ACCESS.2022.3228535.

[45] G. Falazi, U. Breitenbücher, F. Leymann, S. Schulte, Cross-chain smart contract invocations: A systematic multi-vocal literature review 56 (2024). URL: https://doi.org/10.1145/3638045. doi:10.1145/3638045.

[46] The Solidity Authors, Solidity documentation | contracts | events, 2024. https://docs.soliditylang.org/en/v0.8.28/contracts.html#events Accessed 10/2024.

[47] A. Bigiotti, L. Mostarda, A. Navarra, A. Pinna, R. Tonelli, M. Vaccargiu, Interoperability between evm-based blockchains, in: L. Barolli (Ed.), Advanced Information Networking and Applications, Springer Nature Switzerland, Cham, 2024, pp. 98–109.

[48] Web3 Labs, Web3j, 2024. https://docs.web3j.io Accessed 09/2024.

[49] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, J. Yellick, Hyperledger fabric: a distributed operating system for permissioned blockchains, in: Proceedings of the Thirteenth EuroSys Conference, EuroSys '18, Association for Computing Machinery, New York, NY, USA, 2018. URL: https://doi.org/10.1145/3190508.3190538. doi:10.1145/3190508.3190538.

[50] L. Olivieri, V. Arceri, B. Chachar, L. Negrini, F. Tagliaferro, F. Spoto, P. Ferrara, A. Cortesi, General-purpose languages for blockchain smart contracts development: A comprehensive study, IEEE Access 12 (2024) 166855–166869. doi:10.1109/ACCESS.2024.3495535.

[51] Goodman, LM, Tezos: A self-amending crypto-ledger, 2014. White paper. https://tezos.

com/whitepaper.pdf. Accessed 10/2024.
[52] B. Jeannet, A. Miné, Apron: A library of numerical abstract domains for static analysis, in: International Conference on Computer Aided Verification, Springer, 2009, pp. 661–667.
[53] G. Costantini, P. Ferrara, A. Cortesi, A suite of abstract domains for static analysis of string values, Softw. Pract. Exper. 45 (2015) 245–287. URL: https://doi.org/10.1002/spe.2218. doi:10.1002/spe.2218.
[54] P. Ferrara, L. Negrini, Sarl: Oo framework specification for static analysis, in: M. Christakis, N. Polikarpova, P. S. Duggirala, P. Schrammel (Eds.), Software Verification, Springer International Publishing, Cham, 2020, pp. 3–20.
[55] C. Schneidewind, M. Scherer, M. Maffei, The good, the bad and the ugly: Pitfalls and best practices in automated sound static analysis of ethereum smart contracts, in: T. Margaria, B. Steffen (Eds.), Leveraging Applications of Formal Methods, Verification and Validation: Applications, Springer International Publishing, Cham, 2020, pp. 212–231.
[56] Z. Lv, D. Wu, W. Yang, L. Duan, Attack and protection schemes on fabric isomorphic crosschain systems, International Journal of Distributed Sensor Networks 18 (2022) 15501477211059945.
[57] P. Ferrara, L. Olivieri, F. Spoto, Static privacy analysis by flow reconstruction of tainted data, International Journal of Software Engineering and Knowledge Engineering 31 (2021) 973–1016. URL: https://doi.org/10.1142/S0218194021500303. doi:10.1142/S0218194021500303.
[58] P. Ferrara, L. Olivieri, F. Spoto, Tailoring taint analysis to gdpr, in: M. Medina, A. Mitrakas, K. Rannenberg, E. Schweighofer, N. Tsouroulas (Eds.), Privacy Technologies and Policy, Springer International Publishing, Cham, 2018, pp. 63–76. URL: https://doi.org/10.1007/978-3-030-02547-2_4. doi:10.1007/978-3-030-02547-2_4.