



GoLiSA: An Advanced Static Analysis Tool for Detecting Potential Risks and Vulnerabilities in Hyperledger Fabric Smart Contracts

Luca Olivieri[✉]

Ca' Foscari University of Venice
Venice, Italy
luca.olivieri@unive.it

Luca Negrini

Ca' Foscari University of Venice
Venice, Italy
luca.negrini@unive.it

Vincenzo Arceri

University of Parma
Parma, Italy
vincenzo.arceri@unipr.it

Pietro Ferrara

Ca' Foscari University of Venice
Venice, Italy
pietro.ferrara@unive.it

Fausto Spoto

University of Verona
Verona, Italy
fausto.spoto@univr.it

Agostino Cortesi

Ca' Foscari University of Venice
Venice, Italy
cortesi@unive.it

Abstract

GoLiSA is a static analysis tool that allows the verification of smart contracts written in Go, the most widely used language for developing smart contracts in Hyperledger Fabric. GoLiSA relies on abstract interpretation to model program semantics and identify potential risks and vulnerabilities in smart contracts. It supports advanced analyses such as information flow tracking and numerical and string value analysis, enabling the approximation of semantic program behaviors. Building on these analyses, GoLiSA implements a suite of checks for detecting non-trivial issues and potential risks in Hyperledger Fabric, including non-determinism, phantom reads, untrusted cross-contract invocations, read-after-write and overwrite issues, cross-channel invocation risks, numerical issues, and unhandled errors.

GoLiSA is publicly available at <https://github.com/lisa-analyzer/go-lisa>. A demonstration video showcasing the tool is available at <https://youtu.be/WqkYOpWc9oE>.

CCS Concepts

• Security and privacy → Distributed systems security; • Computer systems organization → Peer-to-peer architectures; Dependable and fault-tolerant systems and networks; • Information systems → Distributed storage.

Keywords

Hyperledger Fabric, Static Analysis, Smart Contract, Blockchain, Chaincode, Abstract Interpretation, Program Verification

ACM Reference Format:

Luca Olivieri, Luca Negrini, Vincenzo Arceri, Pietro Ferrara, Fausto Spoto, and Agostino Cortesi. 2026. GoLiSA: An Advanced Static Analysis Tool for Detecting Potential Risks and Vulnerabilities in Hyperledger Fabric Smart Contracts. In *Proceedings of the 41st IEEE/ACM International Conference on Automated Software Engineering (ASE '26)*, October 12–16, 2026, Munich, Germany. ACM, New York, NY, USA, 5 pages. <https://doi.org/10.1145/3832783.3834644>



This work is licensed under a Creative Commons Attribution 4.0 International License. ASE '26, Munich, Germany

© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2882-2/2026/10
<https://doi.org/10.1145/3832783.3834644>

1 Introduction

Hyperledger Fabric (HF) ¹ is a permissioned blockchain framework for enterprise applications, supported by the LF Decentralized Trust of the Linux Foundation. Its smart contracts, called *chaincode*, can be written in general-purpose programming languages [13], among which Go is the most widely used due to its performance and efficiency. In HF, the *ledger* is composed of two main parts: (i) the *blockchain*, which is an immutable log of ordered transactions, and (ii) the *world state*, a key-value database holding the current mutable state. The world state is accessed and updated by smart contracts through specific key-value operations such as *GetState* and *PutState*, which perform read and write operations, respectively. In addition, HF organizes data and communication through channels, which are private sub-networks of participants. Each channel maintains its own independent ledger and smart contracts, enabling confidentiality and data isolation within the same network.

In this context, security is crucial because HF is primarily designed to maintain ledgers populated by smart contracts that manage industrial processes and track critical data. Furthermore, the development of smart contracts is prone to errors that can introduce various risks and vulnerabilities [9, 27].

This paper presents GoLiSA, a tool for the static analysis and verification of HF smart contracts written in Go. GoLiSA relies on abstract interpretation [4] to model program semantics and identify potential risks and vulnerabilities. It supports several advanced analyses [12, 15–18, 20], enabling the detection of a wide range of issues prior to deployment, which is especially critical in blockchain environments where data becomes immutable once smart contracts are executed. GoLiSA targets both practitioners (chaincode developers and security engineers) who need a pre-deployment auditing tool, and researchers working on static analysis or blockchain security, who can leverage its extensible architecture to prototype and evaluate new analyses. It operates directly on Go source code and can be integrated as a standalone analysis step in development pipelines.

2 GoLiSA

GoLiSA builds on LiSA [7, 10, 11], a library for static analysis based on abstract interpretation [4]. LiSA provides the core infrastructure for building analyzers, including control-flow graph (CFG)

¹Hyperledger Fabric Documentation: <https://hyperledger-fabric.readthedocs.io/en/latest/whatis.html> (Accessed 04/2026)

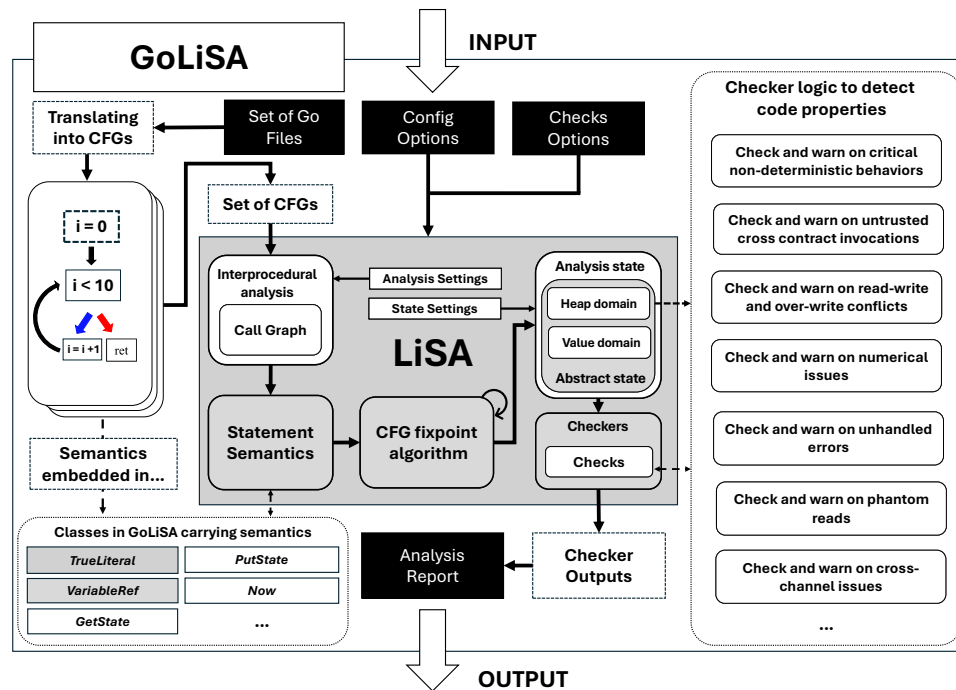


Figure 1: The architecture of GoLiSA.

representations, fixpoint algorithms, and a modular architecture for composing analysis components.

Fig. 1 shows the overall architecture: gray components denote LiSA core infrastructure, while white components represent GoLiSA-specific extensions that customize LiSA for analyzing Go programs.

The analysis workflow of GoLiSA begins with the translation of Go source code into LiSA CFGs (Fig. 1) and the definition of Go-specific semantics, types, and execution rules (e.g., variable scoping and shadowing). These CFGs are then analyzed by LiSA in a language-independent manner. In particular, they are processed by an interprocedural analysis that cooperates with a call graph to resolve function calls and propagate results across procedures.

Fixpoints are computed over CFGs, following a chosen analysis strategy (e.g., modular or call-chain-based). Each CFG node is translated into symbolic expressions that model the low-level effects of executing program statements. These expressions are evaluated over an abstract state [6], which combines a heap domain (to approximate memory structures) and value domains (e.g., numerical, string, or information-flow domains) to track program values.

The abstract state computes approximations of program behavior, and the resulting information is used by semantic checks to report potential issues. All components, interprocedural analysis, call graph, abstract state, and domains, are configurable within GoLiSA, enabling a flexible and extensible analysis design.

3 Detection of Issues in HF Smart Contracts

In order to better understand potential risks in HF chaincode, this section provides minimal code snippets that show representative vulnerabilities and demonstrates how GoLiSA detects them.

Non-determinism [20]. Blockchain systems require deterministic execution so that all nodes produce the same result when committing a transaction, but general-purpose languages can introduce non-determinism through features like system time, randomness, or concurrency, potentially causing inconsistent outputs and consensus failures.

Since some forms of non-determinism may be intentional or beneficial, GoLISA focuses only on those that are harmful: it tracks how non-deterministic values propagate through the program via information flow analysis and raises a warning when such values influence critical operations such as state updates. For example, consider the following code:

```
1 t := time.Now()
2 value := []byte(t.String())
3 err := stub.PutState(key, value)
```

GoLiSA recognizes `time.Now()` as a source of non-deterministic values because system local time may differ across peers. Through information flow analysis, it tracks how this value propagates from `t` into `value` and then into the critical function `PutState`, which writes data to the ledger. GoLiSA raises a warning because a non-deterministic value affects an operation affecting the state, thus leading to possible consensus failures.

Phantom reads [16]. In HF, phantom reads are partially mitigated through read-set validation and selective range-query re-execution, but the optimistic concurrency model may degrade performance under high contention. Moreover, some range queries are not revalidated, potentially leaving phantom reads undetected when their results affect ledger updates.

GoLiSA applies information flow analysis to detect potential phantom read conflicts by tracking how data from range queries, as sources of phantom data, propagates through the program and reaches critical write operations. For example, consider the following code:

```
1 it, err = stub.GetStateByRange("A", "Z")
2 // ...
3 kv, err = it.Next()
4 // ...
5 v = kv.Value
6 err = stub.PutState(mykey, v)
```

GoLiSA considers `GetStateByRange` as a source, and propagates information into `kv.Value`, then into `v`. This value flows into `PutState`, meaning that a write operation depends on data that may change due to concurrent modifications of the queried range, indicating a potential phantom read. GoLiSA raises a warning in this case.

Untrusted cross-contract invocations [18]. The blockchain is a trustless environment, so smart contracts must treat external input as untrusted. If such input affects cross-contract calls, it may be exploited to manipulate execution flow, leading to vulnerabilities such as arbitrary code execution or state corruption.

GoLiSA applies a two-phase information flow analysis: (i) it detects when untrusted data reaches cross-contract invocations, and (ii) it identifies when such invocations result in untrusted writes to blockchain storage. For instance, consider the following code:

```
1 args := stub.GetStringArgs()
2 c := args[0]
3 resp := stub.InvokeChaincode(c, cArgs, "ch1")
4 // ...
5 err := stub.PutState("key", response.Payload)
```

First, GoLiSA performs an information flow analysis starting from `GetStringArgs`, a source of untrusted input, and propagates it through the program. It then detects that this untrusted data reaches the cross-contract invocation `InvokeChaincode` via the `c` variable, which is user-controlled and may arbitrarily change the target contract, enabling unintended executions. Next, GoLiSA treats the identified untrusted cross-contract invocations as additional sources in a second analysis pass, propagating information again through the program. Finally, GoLiSA detects that data originating from these invocations flows into the write operation `PutState`, raising warnings for both untrusted cross-contract invocations and untrusted data storage.

Read-after-write and over-write issues [17]. HF does not support *read-your-writes* semantics. This leads to two subtle issues: (i) *read-after-write*, where reads return only the committed state and ignore prior uncommitted writes to the same key; and (ii) *over-write*, where multiple writes to a key result in only the most recently executed value being committed.

To detect these issues, GoLiSA over-approximates possible key values via string value analysis, then identifies potential conflicts for each read-write and write-write pair. If an execution path can trigger both operations, it raises a warning. GoLiSA also accounts for programs involving `defer` instructions. For example, consider the following fragment:

```
1 err := stub.PutState("A", []byte(v))
2 // ...
3 r, err = stub.GetState("A")
4 // ...
5 err = stub.PutState("A", []byte(z))
```

GoLiSA detects that `PutState` and `GetState` operate on the same key ("A"). It constructs both the read-write pair (lines 1 and 3) and the write-write pair (lines 1 and 5), then traverses the CFG to check their execution order. It identifies that a read may occur after a prior write and that a write may follow another write on the same key, leading to an over-write. In both cases, it triggers a warning.

Cross-channel invocation risks [12]. Cross-contract invocations across different channels can lead to missed writes and reduced transparency, as the target channel's ledger is neither updated nor recorded. GoLiSA detects these issues by inferring channel and contract identifiers via string analysis, then traversing the CFGs to identify potentially unsafe cross-channel invocations. GoLiSA also supports multiple cross-contract analyses to detect more sophisticated scenarios involving write operations (see details in [12]). Consider the following fragment:

```
1 resp1 := stub.InvokeChaincode(c, cArgs, "chA")
2 // ...
3 resp2 := stub.InvokeChaincode(c, cArgs, "chB")
```

GoLiSA approximates string values and determines that the two `InvokeChaincode` calls perform cross-contract invocations on different channels, namely "chA" and "chB". If the target contracts and their relevant information are available, GoLiSA further analyzes whether write operations can be executed on those contracts and emits a warning when such operations occur across different channels. Otherwise, if information about the target contracts or the contracts themselves is unavailable, GoLiSA conservatively raises a warning.

Numerical issues. HF inherits numerical issues from Go. Integer wrap-around semantics can cause silent overflow/underflow, while division by zero triggers a runtime panic that aborts execution, leading to transaction failures. GoLiSA detects these issues by over-approximating numerical values. For detecting overflow and underflow, it checks whether they exceed the type's maximum or minimum bounds. For division-by-zero, it verifies whether potential divisors may hold zero. In both cases, GoLiSA triggers a warning. For example, consider the following fragment:

```
1 var x uint8 = 200 + n // n is an input value
2 var y uint8 = 1 / m // m is an input value
```

GoLiSA computes an approximation of the numerical values. It then determines that the addition may overflow when `n` is greater than 55, because the maximum value for `uint8` is 255. In the case of the division, `m` originates from a transaction input, so it is inferred to potentially take any integer value, including zero. Then, in both cases, it triggers a warning.

Unhandled errors [15]. If errors are ignored or not explicitly handled, execution may continue with invalid or incomplete state, leading to inconsistent ledger updates or unexpected behavior. GoLiSA analyzes the CFG to detect unhandled error conditions. It flags cases where errors are discarded using the blank identifier,

left unassigned, or propagated without proper validation. In particular, it identifies situations where error variables are not checked within the same function, violating best practices that require immediate handling or verification after each call. For example, consider the following code:

```
1 err := stub.PutState(key, value)
2 return shim.Success("Ok!")
```

GoLiSA identifies the call to `PutState` and determines that its returned error is assigned to a variable `err`. It then infers that this variable is never used in any conditional check throughout the function. Since the error is not handled before the function completes, GoLiSA raises a warning about an unhandled error value.

4 GoLiSA Evaluation

The analyses presented in Section 3 can be reproduced by following the instructions provided in the artifact [19]. Furthermore, GoLiSA has been evaluated across hundreds of existing smart contracts in several published studies [12, 15, 17, 18, 20], showing promising empirical results also in comparison to state-of-the-art tools.

We summarize below the empirical results reported in prior work, where GoLiSA's individual checkers have been evaluated on benchmarks collected from publicly available GitHub repositories. All experiments are reproducible either through the artifacts accompanying the corresponding publications or by following the instructions provided in the referenced papers. For the detection of *non-determinism* [20], GoLiSA achieves a precision of 63.59%, outperforming publicly available state-of-the-art static analysis tools, that achieve precisions ranging from 22.51% to 33.5%. The evaluation was conducted on a benchmark of 651 unique Hyperledger Fabric chaincodes collected from public GitHub repositories. Existing tools rely primarily on denylisting known sources of non-determinism, such as specific library imports and language constructs. For *untrusted cross-contract invocations* [18], to the best of our knowledge, GoLiSA is the first static analysis tool for Hyperledger Fabric to support the detection of vulnerabilities arising from untrusted cross-contract invocations. On a benchmark comprising 420 chaincodes containing `InvokeChaincode` statements, it achieves a precision of 100%, detecting all such vulnerabilities without producing any false positives. For *read-after-write issues* [17], GoLiSA achieves a precision of 95%. In comparison, publicly available state-of-the-art static analysis tools report precisions ranging from 29.41% to 100%, but recalls between only 5.26% and 13.16%. The evaluation was performed on the same benchmark of 651 unique chaincodes. For *over-write issues* [17], GoLiSA achieves a precision of 70.51% on the same dataset. To the best of our knowledge, GoLiSA is currently the only static analysis tool supporting the detection of over-write issues in Hyperledger Fabric chaincode. For *cross-channel risks* [12], GoLiSA achieves a precision of 64.97% on a benchmark containing 575 unique chaincodes with `InvokeChaincode` statements. To the best of our knowledge, GoLiSA is also the only static analysis tool capable of detecting cross-channel risks arising from multi-contract interactions in Hyperledger Fabric. For *unhandled errors* [15], GoLiSA achieves a precision of 99.26% on a benchmark consisting of 651 unique chaincodes. Across all checkers, GoLiSA achieves a recall of 100%, reflecting the soundness of the underlying abstract interpretation-based approach, which guarantees that

no true vulnerability is missed.² For *phantom reads* [16], no sufficiently representative benchmark is currently available to enable a meaningful empirical evaluation. Instead, the checker for *numerical issues* is still under active investigation. Current efforts focus on improving the precision of the analysis by reducing cascades of warning messages due to numerical overflow propagation, while preserving the soundness guarantees of the underlying approach.

Finally, GoLiSA has also been used in proof-of-concept collaborations with industrial partners [21]. Moreover, GoLiSA has been applied to assess the security of smart contracts generated by popular large language models, including ChatGPT, Claude, DeepSeek, and CodeLlama, analyzing more than 8000 chaincodes and identifying potential risks and issues in approximately 29% of the generated files [14].

5 Related Work

Unlike Ethereum, HF lacks a mature formal verification and security analysis ecosystem. Its modular architecture, endorsement model, and chaincode semantics complicate static analysis, causing existing tools to rely mainly on syntactic or intra-procedural techniques rather than full semantic reasoning.

Existing tools for verifying HF issues include Chaincode Analyzer [8] and Revive^{CC} [26], that extend AST-based analyses with intra-procedural checks but remain largely syntactic and pattern-based. Similarly, Lv et al. [9] and Yamashita et al. [27] broaden issue coverage while relying on comparable pattern-matching techniques. Shah et al. [24] also propose an AST-based approach, but the lack of implementation and detailed detection mechanisms prevents meaningful comparison. HFCCT [22] combines AST analysis with symbolic execution for multiple HF issues. However, their detection remains intra-procedural and does not fully exploit symbolic reasoning. Ebrahimi et al. [5] apply model checking to HF smart contracts, by using the Rebeca [25] modeling language to verify concurrent execution properties.

Finally, there is related work targeting specific issues. For instance, Ren et al. [23] and Chen et al. [1] introduce advanced analyses for detecting read-after-write issues. However, these approaches remain limited in their precision and their ability to reason about value-level information. GoDetect [2] performs intra-procedural static taint analysis on an intermediate static single assignment form and a set of focused checks to detect non-deterministic sources, map-iteration dependence, and unsafe global state access. PDChecker, proposed by Chen et al. [3], detects privacy data leakage vulnerabilities in HF chaincodes, identifying patterns where sensitive private collection data is inadvertently exposed.

6 Conclusion

In this paper, we introduced GoLiSA, a static analyzer designed to support advanced abstract interpretation-based analyses, that are not typically applied in this domain. By releasing the tool as open source, we aim to promote more secure smart contract development.

As future work, we plan to extend the coverage of the Go API and program semantics, improve analyses to reduce false positives,

²Note that the soundness guarantee applies only within the supported analysis scope and assumptions, including the Go semantics, APIs/libraries, and language features modeled and supported by the tool. It does not imply general program correctness or the absence of bugs outside this scope.

develop additional checks for emerging vulnerabilities, including privacy-related issues, and explore cross-chain analysis by leveraging the multi-language capabilities of LiSA [11]. We also intend to broaden support to other Go blockchain frameworks, such as the Cosmos SDK, Tendermint Core and its derivatives (e.g., CometBFT and Ignite), as well as Go-Ethereum.

Data Availability Statement

GoLiSA is part of the LiSA project and is released as an open-source static analysis tool. The project repository is available at: <https://github.com/lisa-analyzer/go-lisa>. For technical documentation and implementation details, see the LiSA website: <https://lisa-analyzer.github.io>. The artifact for reproducing the analyses presented in Section 3 is publicly available on Zenodo [19].

Acknowledgments

Work partially supported by SERICS (PE00000014 - CUP H73C22000-89001) project funded by PNRR NextGeneration EU, PR Veneto FESR 2021–2027, Az. 1.4.1 ID 251382_008101 "SmartCycle – Cybersecurity", (CUP H73C25001340002), and by Regione Veneto: RIR project SATCO (CUP D19J24000750007), and RIR project SUPREME (CUP D19J24000810007).

References

- [1] Donghan Chen, Junxiong Lin, and Zhihui Lu. 2025. Static Analysis Detection of Hyperledger Fabric Read-Write Logic Vulnerability. In *Blockchain and Web3.0 Technology Innovation and Application*. Springer Nature Singapore, 142–152. doi:10.1007/978-981-97-9412-6_13
- [2] Xinyuan Chen, Zhaoyu Su, and Haiping Zhou. 2025. GoDetect: Precise Detection of Go Smart Contract Vulnerabilities Based on Static Single Assignment (SSA). In *2025 5th International Conference on Intelligent Communications and Computing (ICICC)*. 638–642. doi:10.1109/ICICC66840.2025.11199745
- [3] Ziming Chen, Yue Li, Jianbo Gao, Jiashuo Zhang, Ke Wang, Jianbin Hu, Zhi Guan, and Zhong Chen. 2024. Understanding and Detecting Privacy Leakage Vulnerabilities in Hyperledger Fabric Chaincodes. In *2024 IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*. 570–581. doi:10.1109/ISSRE62328.2024.00060
- [4] Patrick Cousot. 2021. *Principles of Abstract Interpretation*. MIT Press. <https://mitpress.mit.edu/9780262044905>
- [5] Elmira Ebrahimi, Ehsan Khamsepanah, Marjan Sirjani, and Siamak Mohammadi. 2023. Model Checking of Hyperledger Fabric Smart Contracts. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. 1–8. doi:10.1109/ETFA54631.2023.10275704
- [6] Pietro Ferrara. 2016. A generic framework for heap and value analyses of object-oriented programming languages. *Theor. Comput. Sci.* 631 (2016), 43–72. doi:10.1016/j.tcs.2016.04.001
- [7] Pietro Ferrara, Luca Negrini, Vincenzo Arceri, and Agostino Cortesi. 2021. Static Analysis for Dummies: Experiencing LiSA. In *Proceedings of the 10th ACM SIGPLAN International Workshop on the State Of the Art in Program Analysis (Virtual, Canada) (SOAP 2021)*. Association for Computing Machinery, New York, NY, USA, 1–6. doi:10.1145/3460946.3464316
- [8] kzhry. 2021. Chaincode Analyzer. <https://github.com/hyperledger-labs/chaincode-analyzer>. Accessed 07/2023.
- [9] Penghui Lv, Yu Wang, Yazhe Wang, and Qihui Zhou. 2021. Potential Risk Detection System of Hyperledger Fabric Smart Contract based on Static Analysis. In *IEEE Symposium on Computers and Communications, ISCC 2021, Athens, Greece, September 5–8, 2021*. IEEE, 1–7. doi:10.1109/ISCC53001.2021.9631249
- [10] Luca Negrini, Vincenzo Arceri, Luca Olivieri, Agostino Cortesi, and Pietro Ferrara. 2024. Teaching Through Practice: Advanced Static Analysis with LiSA. In *Formal Methods Teaching*, Emil Sekerinski and Leila Ribeiro (Eds.). Springer Nature Switzerland, Cham, 43–57. doi:10.1007/978-3-031-71379-8_3
- [11] Luca Negrini, Pietro Ferrara, Vincenzo Arceri, and Agostino Cortesi. 2023. *LiSA: A Generic Framework for Multilanguage Static Analysis*. Springer Nature Singapore, Singapore, 19–42. doi:10.1007/978-981-19-9601-6_2
- [12] Luca Olivieri. 2025. Detection of Cross-Channel Invocation Risks in Hyperledger Fabric. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*. 107–118. doi:10.1109/ISSRE66568.2025.00023
- [13] Luca Olivieri, Vincenzo Arceri, Badaruddin Chachar, Luca Negrini, Fabio Tagliaferro, Fausto Spoto, Pietro Ferrara, and Agostino Cortesi. 2024. General-Purpose Languages for Blockchain Smart Contracts Development: A Comprehensive Study. *IEEE Access* 12 (2024), 166855–166869. doi:10.1109/ACCESS.2024.3495535
- [14] Luca Olivieri, David Beste, Luca Negrini, Lea Schönherr, Antonio Emanuele Cinà, and Pietro Ferrara. 2025. Code Generation of Smart Contracts with LLMs: A Case Study on Hyperledger Fabric. In *2025 IEEE 36th International Symposium on Software Reliability Engineering (ISSRE)*. 239–251. doi:10.1109/ISSRE66568.2025.00034
- [15] Luca Olivieri and Luca Negrini. 2025. Don't Panic: Error Handling Patterns in Go Smart Contracts and Blockchain Software. In *2025 7th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*. 1–9. doi:10.1109/BRAINS67003.2025.11302935
- [16] Luca Olivieri, Luca Negrini, Vincenzo Arceri, Badaruddin Chachar, Pietro Ferrara, and Agostino Cortesi. 2024. Detection of Phantom Reads in Hyperledger Fabric. *IEEE Access* 12 (2024), 80687–80697. doi:10.1109/ACCESS.2024.3410019
- [17] Luca Olivieri, Luca Negrini, Vincenzo Arceri, Pietro Ferrara, and Agostino Cortesi. 2025. Detection of Read-Write Issues in Hyperledger Fabric Smart Contracts. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing (Catania International Airport, Catania, Italy) (SAC '25)*. Association for Computing Machinery, New York, NY, USA, 329–337. doi:10.1145/3672608.3707721
- [18] Luca Olivieri, Luca Negrini, Vincenzo Arceri, Pietro Ferrara, Agostino Cortesi, and Fausto Spoto. 2025. Static Detection of Untrusted Cross-Contract Invocations in Go Smart Contracts. In *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing (Catania International Airport, Catania, Italy) (SAC '25)*. Association for Computing Machinery, New York, NY, USA, 338–347. doi:10.1145/3672608.3707728
- [19] Luca Olivieri, Luca Negrini, Vincenzo Arceri, Pietro Ferrara, Fausto Spoto, and Agostino Cortesi. 2026. GoLiSA: An Advanced Static Analysis Tool for Detecting Potential Risks and Vulnerabilities in Hyperledger Fabric Smart Contracts (Artifact). Version 1.0 (all versions DOI 10.5281/zenodo.21177773). doi:10.5281/zenodo.21177774
- [20] Luca Olivieri, Luca Negrini, Vincenzo Arceri, Fabio Tagliaferro, Pietro Ferrara, Agostino Cortesi, and Fausto Spoto. 2023. Information Flow Analysis for Detecting Non-Determinism in Blockchain. In *37th European Conference on Object-Oriented Programming (ECOOP 2023) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 263)*, Karim Ali and Guido Salvaneschi (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 1–25. <https://doi.org/10.4230/LIPIcs.ECOOP.2023.23>. doi:10.4230/LIPIcs.ECOOP.2023.23
- [21] Luca Olivieri, Fabio Tagliaferro, Vincenzo Arceri, Marco Ruaro, Luca Negrini, Agostino Cortesi, Pietro Ferrara, Fausto Spoto, and Enrico Talin. 2022. 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 (San Diego, CA, USA) (SOAP 2022)*. Association for Computing Machinery, New York, NY, USA, 23–29. doi:10.1145/3520313.3534658
- [22] PerryLee69. 2021. HFCCT. <https://github.com/PerryLee69/HFCCT>. Accessed 04/2025.
- [23] Feixiang Ren and Sujuan Qin. 2023. Fabric Smart Contract Read-After-Write Risk Detection Method Based on Key Methods and Call Chains. In *Smart Computing and Communication*. Springer Nature Switzerland, Cham, 381–392. doi:10.1007/978-3-031-28124-2_36
- [24] Neelkumar K. Shah, Sachin Nandurkar, Mayur Bilapate, Mohammad Aziz Maalik, Niteshkumar Harné, Karimullah Shaik, and Ajai Kumar. 2023. Smart Contract Vulnerability Detection Techniques for Hyperledger Fabric. In *2023 IEEE 8th International Conference for Convergence in Technology (I2CT)*. 1–7. doi:10.1109/I2CT57861.2023.10126362
- [25] Marjan Sirjani, Ali Movaghar, Amin Shali, and Frank S. de Boer. 2004. Modeling and Verification of Reactive Systems using Rebeca. *Fundam. Inf.* 63, 4 (Dec. 2004), 385–410. doi:10.3233/FUN-2004-63405
- [26] sivachokkapu. 2021. ReviveCC. <https://github.com/sivachokkapu/revive-cc>. Accessed 07/2023.
- [27] Kazuhiro Yamashita, Yoshihide Nomura, Ence Zhou, Bingfeng Pi, and Sun Jun. 2019. Potential Risks of Hyperledger Fabric Smart Contracts. In *2019 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. 1–10. doi:10.1109/IWBOSE.2019.8666486

Received 2026-05-12; accepted 2026-06-19