

A discrete mathematical framework for security and cryptographic analysis of Ethereum smart contracts

Satpal Singh Kushwaha [@]

Department of Computer Science and Engineering

Manipal University Jaipur

Jaipur 303007

Rajasthan

India

Sonam Gupta ⁺

Department of Computer Science and Engineering

Ajay Kumar Garg Engineering College

Ghaziabad 201015

Uttar Pradesh

India

Lipika Goel [§]

Department of Computer Science and Engineering (AIML)

Sridevi Women's Engineering College

Hyderabad 500075

Telangana

India

Abhay Kumar Agarwal [†]

Department of Computer Science and Engineering

Kamla Nehru Institute of Technology

Sultanpur 228118

Uttar Pradesh

India

[@] E-mail: satpal.singh@jaipur.manipal.edu

⁺ E-mail: guptasonam6@gmail.com

[§] E-mail: lipika.bose@gmail.com

[†] E-mail: abhay.knit08@gmail.com

Prashant Hemrajani *

Department of Internet of Things & Intelligent Systems

Manipal University Jaipur

Jaipur 303007

Rajasthan

India

Abstract

Ethereum smart contracts execute immutable and financially critical logic, making post-deployment security flaws especially expensive to remediate. Existing protection mechanisms typically depend on either static code inspection or dynamic execution-based testing, often lacking formal security assurances. This paper introduces a discrete mathematics based hybrid static and dynamic security and encryption analysis framework for Ethereum smart contracts that is both formally rigorous and practically deployable. Smart contracts are represented as discrete transition systems composed of finite states, executable actions, and security invariants, enabling formal reasoning about contract behavior. Static analysis derives control-flow and state-transition relationships to identify invariant violations before deployment, while dynamic analysis validates runtime execution traces against the discrete mathematical model. The framework additionally incorporates an encryption aware access control mechanism defined through key-dependent discrete transitions. Experimental evaluation on vulnerable Ethereum smart contracts shows enhanced vulnerability detection coverage and a reduction in false positives compared to standalone analysis approaches. The proposed framework provides a scalable, mathematically grounded solution for improving the security, encryption assurance, and reliability of Ethereum smart contracts.

Subject Classification: 68M25, 68N30.

Keywords: *Ethereum, Smart contracts, Discrete mathematics, Static analysis, Dynamic testing, Blockchain security, Innovation.*

1. Introduction

Ethereum smart contracts execute programmable logic for decentralized applications but are prone to vulnerabilities like reentrancy, access-control flaws, and arithmetic errors, leading to major financial losses. Since contracts are immutable after deployment, thorough pre- and post-deployment security analysis is critical [1]. Existing approaches use either static analysis or dynamic testing [2], [3], but often lack formal guarantees and suffer from false positives or limited coverage. To address this, we propose a discrete mathematics-based hybrid framework that models contracts as transition systems and integrates static and dynamic analysis for more effective and formally grounded security reasoning.

* E-mail: prashant.hemrajani@jaipur.manipal.edu (Corresponding Author)

1.1 *Motivation*

Ethereum smart contracts execute immutable logic controlling financial assets, making any deployment vulnerability permanently exploitable. Despite existing tools, attacks persist due to limited coverage and lack of formal guarantees—static analysis yields false positives, while dynamic testing misses execution paths. Many tools rely on heuristics instead of rigorous models. Since Ethereum contracts have finite states and deterministic transitions, they are ideal for discrete mathematical modeling, motivating a unified framework that combines static, dynamic, and encryption analysis to improve accuracy and enable provable security guarantees.

1.2 *Contributions*

We model Ethereum smart contracts as finite-state transition systems with defined relations and security invariants, enabling formal reasoning about behavior. A unified framework combines static analysis and dynamic testing through a shared discrete abstraction, where vulnerabilities are defined as invariant violations. Cryptographic authorization is incorporated via key-dependent transitions to strengthen access control. Evaluation on vulnerable contracts shows improved coverage and fewer false positives than standalone methods.

1.3 *Outline*

The paper is organized as follows: Section 2 reviews related work on smart contract security; Section 3 presents the proposed methodology; Section 4 provides experimental results and comparisons; Section 5 discusses limitations and future work; and Section 6 concludes.

2. **Literature review**

Smart contract security analysis includes static/symbolic analysis, dynamic testing/fuzzing, and formal verification, each addressing different vulnerabilities but lacking complete guarantees alone. Early tools like Oyente [4] used symbolic execution but faced path explosion and false positives. Slither [5] improved scalability via intermediate representations, while Mythril [6] extended symbolic and concolic analysis to EVM bytecode. Pattern-based tools such as SmartCheck [7] and Securify [8] use predefined rules and semantic patterns for vulnerability detection. Dynamic approaches include ContractFuzzer [9], which uses ABI-guided fuzzing, and Echidna [10], which applies property-based testing. Formal methods like ZEUS [11] and MAIAN [12] provide stronger guarantees through abstract interpretation and trace analysis. However, these methods largely operate independently without a unified discrete mathematical abstraction, motivating the proposed hybrid framework that integrates static, dynamic, and encryption analysis to improve coverage, precision, and interpretability.

3. Proposed Methodology

This section introduces the proposed discrete mathematics based hybrid static and dynamic security and encryption analysis framework for Ethereum smart contracts.

3.1 Discrete Mathematical Model of Smart Contract

The proposed framework models a smart contract as a discrete transition system. Formally, a smart contract is represented as $\mathcal{M} = (S, A, \delta, \mathcal{I})$ where $S = \{s_0, s_1, \dots, s_n\}$ denotes a finite set of contract states, $A = \{a_1, a_2, \dots, a_m\}$ represents a finite set of externally callable contract actions, and $\delta: S \times A \rightarrow S$ is a state transition function derived from Ethereum Virtual Machine execution semantics. The set $\mathcal{I} = \{I_1, I_2, \dots, I_k\}$ denotes a collection of security invariants. Each invariant $I_j: S \rightarrow \{0,1\}$ encodes a safety property such as access-control correctness, balance consistency, or reentrancy freedom. A contract execution is considered secure if and only if:

$$\forall s \in S, \forall I_j \in \mathcal{I}, I_j(s) = 1 \quad (1)$$

Any transition $(s, a, s') \in \delta$ such that $\exists I_j \in \mathcal{I}$ with $I_j(s') = 0$ constitutes a security violation. Invariant violations detected by either static or dynamic components are propagated to the reporting engine shown in Figure 1.

3.2 Static Analysis Engine

The Static Analysis Engine analyzes the smart contract source code prior to execution. It constructs a control-flow graph (CFG) defined as $G = (V, E)$, where $V \subseteq S$ represents abstract program states and $E \subseteq V \times V$ represents possible execution paths induced by function calls and control statements. Vulnerabilities such as reentrancy are identified as unsafe cycles in the CFG. Formally, a reentrancy vulnerability exists if:

$$\exists C \subseteq G \text{ s.t. } \exists v \in C, I_{\text{reentrancy}}(v) = 0 \quad (2)$$

Similarly, access-control violations and arithmetic flaws correspond to transitions that violate their respective invariants. The static analysis engine outputs a set of potential invariant violations $V_s = \{v \in V \mid \exists I_j \in \mathcal{I}, I_j(v) = 0\}$ which are forwarded to the hybrid fuzzing stage for further validation.

3.3 Hybrid Fuzzing and Dynamic Analysis

The Hybrid Fuzzing Engine combines guided input generation with execution feedback to explore runtime behaviors that may not be captured statically. Dynamic execution of the smart contract produces finite execution traces of the form:

$$\tau = \langle s_0, a_1, s_1, a_2, \dots, a_n, s_n \rangle \quad (3)$$

where each $s_i \in S$ and $a_i \in A$. Each trace represents a discrete path through the transition system $\mathcal{M}$. Trace validation is performed by checking invariant preservation $\forall s_i \in \tau, \forall I_j \in \mathcal{I}, I_j(s_i) = 1$. If a trace violates any invariant, it serves as a concrete counterexample demonstrating an exploitable execution

path. These violating traces are passed to the taint-based analysis module for deeper inspection.

3.4 Taint-Based Runtime Analysis and Reporting

The Taint-Based Analysis Module tracks sensitive variables such as balances, ownership flags, and authorization states during execution. Taint propagation is modeled as a discrete relation $T \subseteq S \times S$ where $(s_i, s_j) \in T$ indicates that tainted data originating in state s_i influences state s_j . Unauthorized data flows are detected when tainted variables affect critical state transitions that violate security invariants. Formally, a taint violation occurs if:

$$\exists (s_i, s_j) \in T \text{ s.t. } \exists I_k \in \mathcal{I}, I_k(s_j) = 0 \quad (4)$$

Finally, the Reporting Engine consolidates invariant violations identified through static analysis, dynamic trace validation, and taint propagation into a comprehensive vulnerability report.

4. Result and Discussion

This section evaluates the proposed framework through comparative analysis with existing smart contract security tools.

4.1 Comparative analysis with existing tools

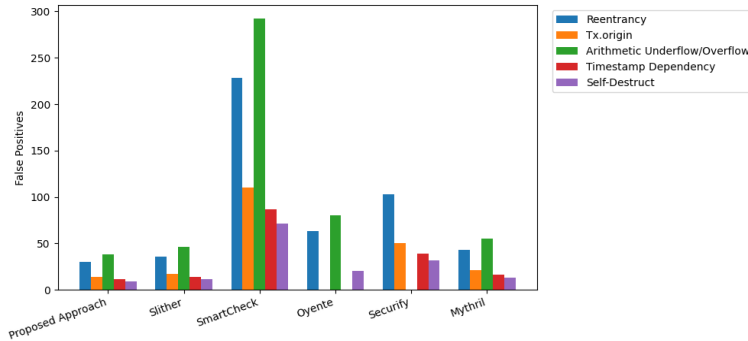


Figure 1
Comparison results for False Positive

We assembled a dataset consisting of 512 vulnerable Ethereum smart contracts written in Solidity, comprising 1,187 annotated vulnerabilities. Among these, 238 contracts were obtained from well-established public benchmarks, including SolidiFI [13], SB Curated [14], and SmartBugs [15], while the remaining 274 contracts were collected through the Etherscan API [16]. The Figure 1 compares false positives reported across major vulnerability classes. Existing static analysis tools report a high number of false positives, particularly for arithmetic and reentrancy vulnerabilities, due to conservative approximations. The proposed framework consistently produces fewer false positives by validating static findings through dynamic execution and discrete

invariant checking. The Figure 2 shows the comparison of detected versus undetected vulnerabilities. The proposed framework achieves the highest detection rate, identifying a larger proportion of vulnerabilities than existing tools. This improvement is attributed to the hybrid integration of static analysis, fuzzing-based dynamic execution, and discrete mathematical invariant validation. a hybrid design guided by discrete mathematical modeling.

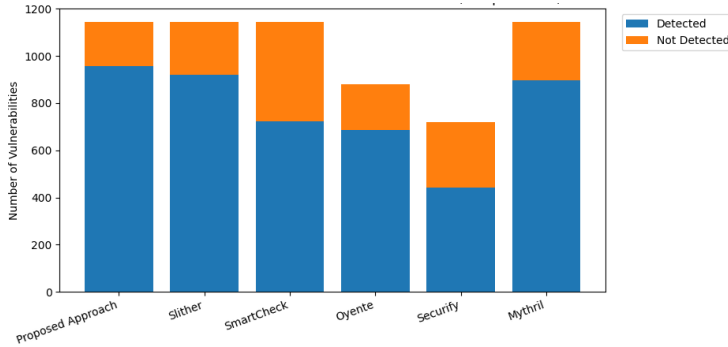


Figure 2
Comparison results for detected and not detected vulnerabilities

4.2 Effectiveness of Discrete Mathematical Integration

The effectiveness of the proposed framework stems from its use of discrete invariants as a unifying abstraction for both static and dynamic analysis. Let V_s denote the set of vulnerabilities identified statically and V_d denote the set of violations observed during dynamic execution. The final vulnerability set is defined as:

$$V = (V_s \cap \text{Reachable}(G)) \cup V_d \quad (5)$$

where $\text{Reachable}(G)$ represents the set of reachable states in the control-flow graph G . This formulation filters unreachable static warnings and incorporates dynamically observed exploit traces, thereby reducing false positives and improving detection confidence.

5. Limitations and Future Research Directions

The comparative results demonstrate that integrating static and dynamic analysis under a discrete mathematical framework significantly improves coverage and interpretability. The proposed framework enables early detection of vulnerabilities while validating exploit feasibility through runtime traces. However, increased state-space complexity may lead to higher analysis overhead for large contracts. Additionally, while fuzzing improves coverage, exhaustive exploration of all execution paths remains infeasible. These results indicate a trade-off between precision and scalability, motivating future work on invariant optimization and state-space reduction.

6. Conclusion

This paper presented a discrete mathematics based hybrid static and dynamic analysis framework for Ethereum smart contracts. By modeling contract behavior as a discrete transition system with finite states and security invariants, the framework enables formal and consistent security analysis. Integrating static analysis, hybrid fuzzing, and runtime trace validation under a unified mathematical abstraction improves vulnerability coverage compared to isolated techniques. Comparative evaluation with existing tools shows that the proposed framework provides broader coverage by combining static analysis, dynamic execution, formal invariant checking, and fuzzing. While scalability challenges remain for large state spaces, future work will focus on automated invariant generation, state-space optimization, and enhanced verification techniques.

References

- [1] V. P. Singh, S. S. Biswas, B. Alankar, and S. Tanweer, "Cryptographic modeling and discrete structures for intrusion detection in Ethereum smart contracts," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 28, no. 8, pp. 3039–3048 (2025).
- [2] A. Dadhich, B. Keswani, and D. Goyal, "Comparative analysis of a novel smart contract-based hybrid access control model for blockchain-enabled secure IoT home automation systems," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 28, no. 7, pp. 2875–2888 (2025).
- [3] V. Ramachandran, H. M. A. Ghanimi, B. Marapelli, N. Kaur, M. R. Laxmi, R. K. Bommisetti, S. Sengan, and P. Dadheech, "An enterprise blockchain model: A reliable cryptography-based cyber-physical systems for securing user data," *Journal of Discrete Mathematical Sciences and Cryptography*, vol. 28, no. 5-B, pp. 2103–2114 (2025).
- [4] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*, Vienna, Austria, pp. 254–269 (2016).
- [5] J. Feist, G. Grieco, and A. Groce, "Slither: A static analysis framework for smart contracts," in *Proc. IEEE/ACM 2nd Int. Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)*, Montreal, QC, Canada, pp. 8–15 (May 2019).
- [6] ConsenSys, "Mythril," GitHub repository. [Online]. Available: <https://github.com/ConsenSys/mythril>. [Accessed: Nov. 5, 2025]

- [7] S. Tikhomirov, E. Voskresenskaya, I. Ivanitskiy, R. Takhaviev, E. Marchenko, and Y. Alexandrov, "SmartCheck: Static analysis of Ethereum smart contracts," in *Proc. 1st Int. Workshop on Emerging Trends in Software Engineering for Blockchain*, Gothenburg, Sweden, pp. 9–16 (May 2018).
- [8] P. Tsankov, A. Dan, D. Drachsler-Cohen, A. Gervais, F. Bünzli, and M. Vechev, "Securify: Practical security analysis of smart contracts," in *Proc. ACM SIGSAC Conf. Computer and Communications Security (CCS)*, Toronto, ON, Canada, pp. 67–82 (2018).
- [9] B. Jiang, Y. Liu, and W. K. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proc. 33rd ACM/IEEE Int. Conf. Automated Software Engineering (ASE)*, Montpellier, France, pp. 259–269 (Sep. 2018).
- [10] G. Grieco, W. Song, A. Cygan, J. Feist, and A. Groce, "Echidna: Effective, usable, and fast fuzzing for smart contracts," in *Proc. 29th ACM SIGSOFT Int. Symp. Software Testing and Analysis (ISSTA)*, Virtual Event, USA, pp. 557–560 (Jul. 2020), doi: 10.1145/3395363.3404366.
- [11] S. Kalra, S. Goel, M. Dhawan, and S. Sharma, "Zeus: Analyzing safety of smart contracts," in *Proc. Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, pp. 1–12 (Feb. 2018).
- [12] I. Nikolić, A. Kolluri, I. Sergey, P. Saxena, and A. Hobor, "Finding the greedy, prodigal, and suicidal contracts at scale," in *Proc. 34th Annual Computer Security Applications Conference (ACSAC)*, San Juan, PR, USA, pp. 653–663 (Dec. 2018).
- [13] SmartBugs Project, "SolidiFI Benchmark," *GitHub repository*. [Online]. Available: <https://github.com/smartbugs/SolidiFI-benchmark>. [Accessed: Nov. 5, 2025].
- [14] SmartBugs Project, "SB Curated: A curated dataset of vulnerable Solidity smart contracts," *GitHub repository*. [Online]. Available: <https://github.com/smartbugs/smartbugs-curated>. [Accessed: Nov. 5, 2025].
- [15] SmartBugs Project, "SmartBugs Wild Dataset," *GitHub repository*. [Online]. Available: <https://github.com/smartbugs/smartbugs-wild>. [Accessed: Nov. 5, 2025].
- [16] Etherscan, "The Ethereum Blockchain Explorer." [Online]. Available: <https://etherscan.io>. [Accessed: Nov. 5, 2025].

Received December, 2025