Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Formal Verification of Smart Contracts for EEG Data Governance: A Case Study with Slither and Formal Specification

Jonathas Tavares Neves

jonathas.neves@ufam.edu.br

Federal University of Amazonas

Moisés Pereira Bastos

Federal University of Amazonas

Lucas Carvalho Cordeiro

University of Manchester

Carlos Augusto de Moraes Cruz

Federal University of Amazonas

Research Article

Keywords: Formal verification, Smart contracts, Blockchain, EEG, Static analysis, Slither, Mythril, IoT security, Medical data governance, Solidity

Posted Date: June 29th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-9805828/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Formal Verification of Smart Contracts for EEG Data Governance Using Slither Static Analysis and Formal Specification

Jonathas Tavares Neves Moisés Pereira Bastos Lucas Carvalho Cordeiro Carlos Augusto de Moraes Cruz

J. T. Neves, M. P. Bastos: Graduate Program in Electrical Engineering (PPGEE), Federal University of Amazonas (UFAM), Manaus, AM 69067-005, Brazil
jonathas.neves@ufam.edu.br; moises.bastos@ufam.edu.br

L. C. Cordeiro: Department of Computer Science, University of Manchester, Manchester M13 9PL, UK; and PPGEE/PPGI, UFAM, Manaus, AM 69067-005, Brazil
lucascordeiro@ufam.edu.br

C. A. de Moraes Cruz: Department of Electronics and Computing, UFAM, Manaus, AM 69067-005, Brazil
carlosamcruz@ufam.edu.br

Received: 23 November 2024 / Accepted: 4 December 2024

Abstract

As artificial intelligence (AI) increasingly powers electroencephalography (EEG)-based brain-computer interface (BCI) systems, the smart contracts that govern their blockchain data pipelines become critical safety boundaries that must be formally verified. This paper presents a formal verification case study of smart contracts for EEG data governance, combining Hoare-logic-based formal specification with two state-of-the-art automated tools: Mythril (SMT-based symbolic execution) and Slither (static analysis). We evaluate all three approaches on contracts integrated with the Kara-One imagined-speech EEG dataset (406 records, 32 channels, 6 participants) using a controlled mutation approach: an array out-of-bounds vulnerability is deliberately seeded into the `checkFinal()` function to assess each tool's detection capability under known conditions. Formal specification successfully identifies the seeded bug through systematic precondition analysis (1/1 targeted vulnerabilities detected; 0 false positives), while both Slither and Mythril fail to detect it—demonstrating a fundamental limitation of automated tools for logical correctness bugs that require precondition reasoning. Complementarily, both automated tools correctly detect three HIGH-severity known vulnerability patterns (reentrancy, controlled delegatecall, unprotected selfdestruct) in validation contracts. We successfully verify eight safety properties including record count limits, array bounds, and data integrity constraints. The results demonstrate that formal specification is essential for safety-critical AI-enabled blockchain-IoT medical applications, providing correctness guarantees that purely automated analysis cannot achieve. Our defense-in-depth strategy combines rigorous formal methods with complementary automated tools for comprehensive smart contract security in AI-driven BCI systems.

Keywords: Formal verification · Smart contracts · Blockchain · EEG · Static analysis · Slither · Mythril · IoT security · Medical data governance · Solidity

1 Introduction

Blockchain technology has emerged as a promising solution for secure and transparent data governance in medical Internet of Things (IoT) applications. Electroencephalography (EEG) devices, increasingly deployed in brain-computer interface (BCI) systems and neurological monitoring, generate sensitive patient data requiring robust access control and integrity guarantees. However, smart contracts—the self-executing programs governing blockchain transactions—are notoriously vulnerable to critical bugs that can compromise patient data security and system availability.

Recent high-profile incidents demonstrate the severity of smart contract vulnerabilities. In 2016, a reentrancy vulnerability in The DAO was exploited to drain over $50 million in Ether from a $168 million investment pool [1], while a 2017 uninitialized ownership bug in the Parity multi-signature wallet permanently froze approximately 513,774 ETH [2]. In medical applications, such failures could have life-threatening consequences by exposing sensitive patient data or corrupting medical records, making formal verification an essential requirement rather than an optional enhancement.

This paper addresses the critical challenge of ensuring correctness in smart contracts for EEG data governance through comprehensive formal verification combining manual formal specification with automated analysis tools. We employ a *controlled mutation approach*—a standard methodology in the formal verification community [4]—whereby a logical bug is deliberately seeded to assess whether each verification paradigm can detect it. We demonstrate a critical finding: automated static analysis tools like Slither, while excellent for detecting known vulnerability patterns,

fundamentally cannot detect certain classes of logical bugs that require runtime reasoning and precondition validation.

We make the following contributions:

1. **Formal specification framework** for EEG data governance smart contracts with complete preconditions, postconditions, and invariants.
2. **Controlled mutation case study** of three verification approaches (Formal Specification, Mythril, Slither) demonstrating complementary strengths and critical limitations.
3. **Discovery of a critical seeded bug** undetectable by automated tools: array out-of-bounds in `checkFinal()` requiring formal precondition analysis.
4. **Real-world integration** with the Kara-One imagined-speech EEG dataset [3], validating applicability with 406 records across 6 participants, 32 channels, and 19 sessions.
5. **Open-source implementation** enabling reproducibility with a complete verification pipeline.

2 Related Work

2.1 Smart Contract Verification

Formal verification of smart contracts has received significant attention following major security incidents. Luu et al. [5] introduced Oyente, a symbolic execution tool detecting transaction-ordering dependencies and timestamp dependencies. Torres et al. [6] developed Osiris for integer bug detection using taint analysis and SMT solvers. Tsankov et al. [7] proposed Securify, combining automated verification with security patterns expressed in Datalog.

These approaches share fundamental limitations: (1) incomplete path coverage due to state explosion in symbolic execution, (2) inability to verify arbitrary user-defined properties, and (3) focus on known vulnerability patterns rather than logical correctness.

2.2 Static Analysis for Smart Contracts

Slither [8] has become the industry-standard static analysis framework for Solidity, featuring 90+ detectors via AST traversal and data flow analysis. However, its fundamental limitation is the inability to reason about runtime values. As demonstrated in Sect. 4, Slither cannot detect the seeded array out-of-bounds bug because it requires symbolic reasoning about the relationship between function parameters and array bounds.

2.3 Bounded Model Checking for Smart Contracts

Song et al. [9] proposed ESBMC-Solidity, which translates Solidity contracts into ESBMC's intermediate representation [10] for SMT-based bounded model checking, demonstrating superior speed and counterexample generation on targeted benchmarks. Our work takes a complementary approach by directly analyzing Solidity contracts through formal specification and static analysis.

2.4 Medical Data Governance

Blockchain-based medical data management has been explored by Azaria et al. [11] (MedRec), Xia et al. [12] (MeDShare), and Roehrs et al. [13] (OmniPHR). None employ formal verification to guarantee smart contract correctness. Our work bridges formal methods and decentralized healthcare with real-world dataset integration.

3 Methodology

3.1 Formal Specification Framework

For the core `EEGDataGovernanceFormal` contract, we define:

State Variables:

- `MAX_RECORDS`: Maximum array size (10)
- `recordCount`: Current registered records
- `records[]`: Array of EEG record structures

Invariants (must hold after every function execution):

- **I1:** $recordCount \leq MAX_RECORDS$
- **I2:** $\forall i \in [0, recordCount) : records[i].exists = true$
- **I3:** $\forall i \in [0, recordCount) : records[i].recordId = i + 1$
- **I4:** $\forall i \in [0, recordCount) : records[i].timestamp > 0$

Function `checkFinal(uint256 n)` — Preconditions:

- **P1:** $n \leq MAX_RECORDS$ (MISSING — Seeded Bug)
- **P2:** $recordCount = n$

Postconditions:

- **Q1:** $\forall i \in [0, n) : records[i].exists = true$
- **Q2:** $\forall i \in [0, n) : records[i].recordId = i + 1$
- **Q3:** $\forall i \in [0, n) : records[i].timestamp > 0$

The missing precondition P1 constitutes the seeded vulnerability: invoking `checkFinal(11)` with $MAX_RECORDS = 10$ causes an array out-of-bounds access at `records[10]`.

3.2 Verification Workflow

Stage 1: Implementation. Five contract variants in Solidity 0.5.16 and 0.8.17:

1. `EEGDataGovernanceFormal.sol`: seeded bug (25 LOC)
2. `EEG_Optimized.sol`: gas improvements (130 LOC)
3. `EEG_KaraOne.sol`: dataset integration (180 LOC)
4. `EEG_Buggy.sol`: known-pattern vulnerabilities (80 LOC)
5. `BugOverflow.sol`: overflow test (30 LOC)

Stage 2: Manual Formal Analysis. For each contract: identify state-modifying functions; specify preconditions, postconditions, and invariants; verify implementations against specifications; identify missing preconditions.

Stage 3: Automated Tool Analysis. Slither v0.10.0 (static analysis), Mythril v0.24.8 (symbolic execution with Z3 4.12.1), and manual formal specification.

Stage 4: Dataset Integration. 406 Kara-One EEG records registered on-chain with IPFS content hashes.

3.3 Experimental Environment

Intel Core i7-10750H @ 2.60 GHz, 16 GB RAM, 512 GB NVMe SSD; Windows 11 Pro + WSL2 (Ubuntu 22.04 LTS); Python 3.10; Solidity 0.5.16/0.8.17; Node.js 18.x.

3.4 Kara-One Dataset

The Kara-One dataset [3] contains imagined and vocalized phonemic and single-word prompts recorded at the Toronto Rehabilitation Institute, with 64-channel EEG and audio modalities. We register metadata from 406 records covering 6 participants, 32 channels, 19 sessions, and 3 cognitive task categories, with IPFS hashes for data integrity.

4 Results and Discussion

4.1 Critical Bug: Array Out-of-Bounds (Seeded)

The seeded vulnerability in `checkFinal()` was detectable *only* through formal specification, not automated tools.

Vulnerable code:

```
function checkFinal(uint256 n)
    public view {
  // MISSING: require(n<=MAX_RECORDS);
  assert(recordCount == n);
  for (uint256 i = 0; i < n; i++) {
    assert(records[i].exists == true);
    assert(records[i].recordId == i+1);
    assert(records[i].timestamp > 0);
  }
}
```

Table 1: Slither static analysis results

Contract	H	M	L	I	Tot
EEG_mythril	0	0	1	6	7
EEG_Optimized	0	0	0	9	9
BugOverflow	0	0	0	3	3
EEG_KaraOne	0	1	1	7	9
EEG_Buggy	3	0	1	3	7

H=High, M=Medium, L=Low, I=Info

Without precondition P1, `checkFinal(11)` with $MAX_RECORDS = 10$ accesses `records[10]`, which is out of bounds. Slither cannot detect this because it: (1) cannot reason about symbolic parameter values; (2) cannot model runtime relationships between parameters and constants; (3) cannot symbolically execute loops to detect bounds violations; and (4) cannot infer missing preconditions. This is a fundamental limitation rooted in Rice's theorem [4].

4.2 Slither Static Analysis Results

Table 1 presents Slither results across all five contracts. Production contracts exhibit zero HIGH-severity issues. `EEG_Buggy` correctly triggers three HIGH-severity alerts. Average analysis time: 2.1 seconds per contract. Zero issues were reported in `EEGDataGovernanceFormal.sol`, confirming that logical precondition bugs lie outside Slither's detection scope.

The three HIGH-severity detections in `EEG_Buggy.sol`:

1. Reentrancy (HIGH): `withdraw()` calls `msg.sender.call` before updating state, enabling recursive withdrawal.

2. Controlled Delegatecall (HIGH): `execute()` passes a user-supplied address to `delegatecall()`, enabling arbitrary code execution.

3. Unprotected Selfdestruct (HIGH): `destroy()` has no access control, allowing any user to permanently destroy the contract.

4.3 Mythril Symbolic Execution Results

Average analysis time: 45.2 seconds per contract. Production contracts produced no issues. `EEG_Buggy.sol` triggered all three HIGH-severity vulnerabilities (reentrancy, delegatecall injection, unprotected suicide). Like Slither, Mythril reported zero issues in `EEGDataGovernanceFormal.sol`, confirming the seeded logical bug lies outside symbolic execution's default detection scope.

Table 2: Comparative analysis of verification approaches

Method	Seeded OOB	Known (3)	Time
Formal Spec	✓	N/A	∼30 min
Mythril	×	✓ 3/3	45.2 s
Slither	×	✓ 3/3	2.1 s

Table 3: Vulnerability detection by category

Type	Formal	Mythril	Slither
Array OOB	✓	×	×
Missing precondition	✓	×	×
Reentrancy	N/A	✓	✓
Delegatecall	N/A	✓	✓
Selfdestruct	N/A	✓	✓
Code quality	N/A	×	✓

4.4 Formal Specification Results

Detection process:

1. Identify: `checkFinal(uint256 n)`
2. Observe loop: `for (i = 0; i < n; i++)`
3. Note array access: `records[i]`
4. Derive required precondition: $n \leq MAX_RECORDS$
5. Compare with implementation: **MISSING (seeded bug confirmed)**
6. Classify: array out-of-bounds vulnerability

Result: 1/1 targeted vulnerabilities detected; 0 false positives; ∼30 minutes manual review for complete contract suite.

4.5 Comparative Analysis

Tables 2 and 3 compare the three verification approaches.

4.6 Kara-One Dataset Integration

All eight safety properties passed: subject ID range; channel count (32); session range (1–19); task type (3 categories); hash non-nullity; sequential record ID; timestamp monotonicity; array bounds compliance. The single MEDIUM Slither warning is a confirmed false positive.

5 Discussion

5.1 Fundamental Limitations of Automated Analysis

Automated tools have fundamental limitations rooted in Rice's theorem [4]. Static analysis **can**: detect known vulnerability patterns; identify code quality issues; flag suspicious constructs; provide fast CI/CD feedback. It **cannot**: reason about runtime parameter values; infer missing preconditions; prove logical correctness; detect bugs requiring symbolic variable reasoning.

Table 4: Kara-One on-chain registration metrics

Metric	Value
Records registered	406
Participants	6
Channels per record	32
Sessions	19
Cognitive tasks	3
Slither time (s)	2.83
Mythril time (s)	48.1
Formal properties verified	8/8
HIGH issues	0
MEDIUM issues	1*

*False positive: strict enum comparison

5.2 Defense-in-Depth Strategy

Layer 1 — Formal Specification (development): detects logical bugs; proves correctness; zero false positives. Apply to critical functions and safety-critical operations.

Layer 2 — Slither (CI/CD): fast (2.1 s avg); 90+ detectors; low false positive rate. Apply to every commit and pull request.

Layer 3 — Mythril (pre-deployment): detects reentrancy and delegatecall misuse. Apply before deployment and after major changes.

5.3 Recommended Verification Workflow

Stage 1 (Development): specify preconditions/postconditions; run Slither; fix HIGH/MEDIUM issues; verify critical functions manually.

Stage 2 (Pre-Audit): run Mythril; investigate findings; document false positives.

Stage 3 (Deployment): final formal review; verify all preconditions; check invariants; document for compliance.

5.4 Implications for Smart Contract Security

1. Automated tools are necessary but not sufficient. Contracts passing all automated checks can still contain critical vulnerabilities detectable only through formal reasoning.

2. Formal methods must be integrated early. For safety-critical medical applications, formal properties must be specified during design.

3. Tool complementarity requires understanding. Security teams must know when each tool is applicable and what guarantees it provides.

5.5 Limitations and Future Work

Mutation scope. One seeded vulnerability was used. A larger mutation suite would strengthen generalizability.

Gas cost and latency. Future work should measure gas consumption and end-to-end latency; clinical BCI deployment requires sub-300 ms response.

ESBMC integration. ESBMC-Solidity [9] v7.11.0 failed consistently (`ERROR: Please provide a program to verify`) across Solidity 0.5.16, 0.8.0, and 0.8.17. Resolving this is a priority.

Scalability. Contracts are 25–180 LOC. Larger systems (1000+ LOC, interacting contracts) require compositional verification.

Privacy. Integrating zk-SNARKs for privacy-preserving EEG sharing while maintaining correctness guarantees is promising future work.

Automated repair. Automatic insertion of missing preconditions (e.g., `require(n <= MAX_RECORDS)`) is a natural extension.

6 Conclusion

This paper presented a formal verification case study of smart contracts for EEG data governance using a controlled mutation approach. Manual formal specification detects a seeded logical precondition bug (1/1 targeted; 0 false positives) that both Mythril and Slither fail to identify, confirming a fundamental limitation of automated tools: they detect *known patterns* but cannot perform *logical reasoning*. Slither analyzed the contract in 2.1 seconds with zero findings; formal specification immediately identified the missing precondition.

The defense-in-depth approach combines: (1) formal specification for mathematical correctness; (2) Slither for CI/CD feedback; and (3) Mythril for pre-deployment analysis. Integration with the Kara-One dataset (406 records, 32 channels, 6 subjects, 19 sessions) demonstrates real-world applicability, with all eight safety properties verified.

Formal specification is an essential practice—not optional—for smart contracts governing sensitive medical data. Open directions: gas cost evaluation, larger mutation suites, ESBMC integration, and real-time latency benchmarking for clinical BCI systems.

Key takeaways:

- Never rely solely on automated tools for safety-critical contracts.
- Specify formal preconditions for all public functions.
- Integrate Slither in CI/CD (2.1 s overhead).
- Run Mythril before deployment.
- Conduct formal reviews for critical business logic.

Acknowledgements

The authors thank the Federal University of Amazonas (UFAM) and PPGEE for supporting this research. The authors gratefully acknowledge Professor Lucas Carvalho Cordeiro for guidance on formal verification methodologies and Professor Carlos Augusto de Moraes Cruz for insights on cyber-physical systems security and blockchain-IoT applications.

Declarations

Conflict of interest. The authors declare that they have no conflict of interest.

Consent to Participate. Not applicable. This study does not involve human participants, clinical data collection, or any form of direct participant involvement. It is a formal verification case study applied to smart contract code and EEG data governance architecture using a publicly available dataset.

Consent to Publish. Not applicable. No identifiable personal data or participant-generated content is included in this manuscript.

Ethics declaration. Not applicable. This research involves formal verification of software artefacts (smart contracts) and does not involve human subjects, animal subjects, or sensitive personal data requiring ethical approval.

Funding. This work was partially supported by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior — Brasil (CAPES) — Finance Code 001.

Data Availability

The study used the publicly available Kara-One EEG dataset [3], consisting of 406 records across 6 participants and 32 channels, recorded at the Toronto Rehabilitation Institute. The dataset is freely available for academic use and can be accessed at: http://www.cs.toronto.edu/~complingweb/data/karaOne/karaOne.html. Metadata and on-chain registration hashes generated during this study are included in the manuscript. The provided IPFS hashes ensure reproducibility and verification of the on-chain registration process.

References

[1] Mehar MI et al (2019) Understanding a revolutionary and flawed grand experiment in blockchain: The DAO attack. J Cases Inf Technol 21(1):19–32. doi:10.4018/JCIT.2019010102

[2] Parity Technologies (2017) A postmortem on the Parity multi-sig library self-destruct. Technical Report. https://www.parity.io/blog/a-postmortem-on-the-parity-multi-sig-library-self-destru

[3] Zhao S, Rudzicz F (2015) Classifying phonological categories in imagined and articulated speech. In:

Proc IEEE ICASSP, South Brisbane, pp 992–996. doi:10.1109/ICASSP.2015.7178118

[4] Rice HG (1953) Classes of recursively enumerable sets and their decision problems. Trans Amer Math Soc 74(2):358–366

[5] Luu L et al (2016) Making smart contracts smarter. In: Proc ACM CCS, pp 254–269. doi:10.1145/2976749.2978309

[6] Torres CF, Schütte J, State R (2018) Osiris: Hunting for integer bugs in Ethereum smart contracts. In: Proc ACSAC. doi:10.1145/3274694.3274737

[7] Tsankov P et al (2018) Securify: Practical security analysis of smart contracts. In: Proc ACM CCS. doi:10.1145/3243734.3243780

[8] Feist J, Grieco G, Groce A (2019) Slither: A static analysis framework for smart contracts. In: Proc WETSEB. doi:10.1109/WETSEB.2019.00008

[9] Song K, Matulevicius N, de Lima Filho EB, Cordeiro LC (2022) ESBMC-Solidity: An SMT-based model checker for Solidity smart contracts. In: Proc ICSE-Companion, Pittsburgh, pp 65–69. doi:10.1145/3510454.3516855

[10] Cordeiro LC, Fischer B, Marques-Silva J (2012) SMT-based bounded model checking for embedded ANSI-C software. IEEE Trans Softw Eng 38(4):957–974. doi:10.1109/TSE.2011.59

[11] Azaria A et al (2016) MedRec: Using blockchain for medical data access and permission management. In: Proc OBD. doi:10.1109/OBD.2016.11

[12] Xia Q et al (2017) MeDShare: Trust-less medical data sharing among cloud service providers via blockchain. IEEE Access 5:14757–14767. doi:10.1109/ACCESS.2017.2730843

[13] Roehrs A, da Costa CA, da Rosa Righi R (2017) OmniPHR: A distributed architecture model to integrate personal health records. J Biomed Inform 71:70–81. doi:10.1016/j.jbi.2017.05.012