Comparative Evaluation of Large Language Models and ESBMC-Solidity for Smart Contract Verification

Glauco Aguiar
Postgraduate Program in Electrical Engineering
Federal University of Amazonas
Manaus, Brazil
glauco.aguiar@ufam.edu.br

Maria Lima
Postgraduate Program in Electrical Engineering
Federal University of Amazonas
Manaus, Brazil
mallu.sdel14@gmail.com

***Abstract*— The growing complexity of Ethereum smart contracts demands verification tools that combine precision with semantic awareness. While ESBMC-Solidity 7.9 achieves high accuracy (78.75%) and precision (73.1%) in detecting low-level vulnerabilities, it fails to identify critical issues such as reentrancy and denial of service. This study evaluates the effectiveness of four Large Language Models: GPT-4.0, Gemini 2.5, Claude 4.0, and Grok 3 Beta against ESBMC using a benchmark of 50 real-world contracts. LLMs showed complementary strengths, particularly in detecting business logic flaws. The results highlight the benefits of hybrid verification strategies combining formal and AI-based approaches for enhanced smart contract security.**

Keywords—vulnerability verification, Large Language Model, Smart Contracts

I. INTRODUCTION

The rise of smart contracts on public blockchains, especially Ethereum, has brought remarkable technological advancements but also critical security challenges. In 2023 alone, losses resulting from the exploitation of vulnerabilities in smart contracts reached $7.1 billion [6]. In this scenario, traditional tools like ESBMC-Solidity offer high accuracy in detecting technical vulnerabilities but require extensive technical expertise and often fail to identify contextual vulnerabilities, such as flaws in business logic [1][11].

Recently, Large Language Models (LLMs), such GPT-4.0, Gemini 2.5, Claude 4.0 and Grok 3 Beta have emerged as promising alternatives, combining advanced semantic capabilities and adaptability to new attack patterns [2][3][10]. However, it remains unclear how much these tools can effectively surpass or complement traditional methods.

This article comparatively evaluates the effectiveness of leading LLMs against the formal verifier ESBMC-Solidity in identifying vulnerabilities in smart contracts, providing practical analysis on their applicability and limitations.

II. CONTEXT

The expansion of decentralized applications (DApps) on Ethereum has intensified the need for effective security mechanisms. Vulnerabilities in smart contracts, such as reentrancy, overflow, and unauthorized access, continue to be critical issues [1][4]. Traditional formal verification tools, despite being highly accurate, face challenges in identifying complex and logical vulnerabilities, thus creating opportunities for more adaptable and comprehensive methods such as LLMs [2][5][11].

III. PROBLEM

Currently, there is no consensus or conclusive study regarding the capability of LLMs to surpass or complement traditional verifiers like ESBMC-Solidity 7.9, particularly in realistic and representative scenarios with practical constraints such as time, computational resources, and required technical expertise [1][6][12].

IV. MOTIVATION

The economic importance and financial security provided by robust smart contracts justify the continuous search for more effective and accessible methods. Traditional models require specific expertise, making their broad application difficult, whereas LLMs offer potential ease of use, even by less specialized users [2][3][10]. This comparative study is essential for providing clear insights to developers, companies, and researchers, contributing to the secure and sustainable advancement of the blockchain ecosystem.

V. GENERAL OBJECTIVE

To comparatively and practically evaluate the effectiveness of the GPT-4.0, Gemini 2.5, Claude 4.0 and Grok 3 Beta, and the formal verifier ESBMC-Solidity 7.9 in identifying vulnerabilities in smart contracts.

VI. ESPECIFIC OBJECTIVES

- Select a balanced set of approximately 50 real world smart contracts from the ScrawlD dataset [4].
- Systematically apply the GPT-4.0, Gemini 2.5, Claude 4.0 and Grok 3 Beta, and ESBMC-Solidity to identify vulnerabilities.
- Evaluate and compare performance using quantitative metrics: recall, precision, false positive and negative rates, and overall accuracy [6][11].

- Identify strengths, limitations, and ideal application contexts for each evaluated method.

VII. RELATED WORK

The growing relevance of smart contracts in decentralized applications has catalyzed substantial research in automated vulnerability detection. Traditional formal verification tools such as ESBMC-Solidity [1], which rely on satisfiability modulo theories (SMT) and bounded model checking, have demonstrated high precision in identifying technical vulnerabilities, including arithmetic errors, reentrancy, and array bounds violations. However, their dependency on user-defined assertions and limited flexibility in recognizing contextual logic flaws have prompted exploration into alternative approaches [5][12].

Recent efforts have turned to the use of Large Language Models (LLMs), leveraging their ability to understand semantic context and reason over unstructured code. Studies such as those by Ince et al. [2] and Sun et al. [3] introduced LLM-based frameworks capable of identifying business logic vulnerabilities often missed by conventional static analyzers. Tools like GPTScan [3] and LLM-Auditor [6] showed promise in uncovering high-level flaws using zero-shot and prompt-based methods.

Yashavant et al. [4] contributed significantly by releasing the ScrawlID dataset, a comprehensive benchmark that maps real-world smart contracts with labeled vulnerabilities across various categories. This dataset became a valuable resource for reproducible evaluation, supporting both LLM-based and formal verification techniques.

Additional research has explored the fine-tuning of LLMs for domain-specific vulnerability detection. Ma et al. [6] and Pirzada et al. [8] demonstrated that LLMs, when enhanced with prompt engineering or fine-tuning, can improve recall in detecting security weaknesses. Meanwhile, Xiao et al. [9] proposed hybrid models combining retrieval-augmented generation with symbolic execution to increase explainability and performance.

Despite these advances, comparative studies involving multiple LLMs evaluated under a unified methodology and against formal verifiers remain limited. Most existing works focus on individual models, small-scale case studies, or synthetic datasets. This gap underscores the necessity for broader evaluations involving real-world contracts and consistent performance metrics, such as the approach proposed in this study.

VIII. METHODOLOGY

The methodological framework adopted in this study comprises a structured multi-step process aimed at assessing the effectiveness of state-of-the-art Large Language Models (LLMs) in the detection of vulnerabilities in Ethereum smart contracts. The approach integrates dataset selection, automation of preprocessing tasks, systematic model evaluation, and quantitative performance analysis.

A. Dataset Selection and Preprocessing

Initially, the ScrawlID dataset [4] was analyzed to identify smart contracts with known and previously categorized vulnerabilities. From this repository, a balanced subset of 50 real-world Ethereum smart contracts was selected, ensuring diversity in terms of size, complexity, and vulnerability types.

A Python script was developed to interface with the Etherscan API, enabling automated verification of the availability of source code for each selected contract. Only contracts with publicly verifiable source code were retained for analysis.

B. Ground Truth Construction

To establish a reliable baseline for evaluation, a reference label set was constructed by aggregating outputs from leading static analysis tools, including Slither, Mythril, SmartCheck, Oyente, and Osiris. A custom Python script was developed to parse and consolidate these results into a standardized ground truth format. This dataset served as the benchmark for subsequent model comparisons.

C. Vulnerability Detection Using LLMs

Each of the 50 contracts was submitted to a predefined prompt tailored for vulnerability detection using the following LLMs: GPT-4.0, Gemini 2.5, Claude 4.0 Sonnet and Grok 3 Beta.

The same prompt template was applied uniformly across all LLMs to ensure consistency. The vulnerability types considered included Arithmetic Errors (ARTHM), Denial of Service (DOS), Logical Errors (LE), Reentrancy (RENT), Time Manipulation (TimeM), Incorrect Time Usage (TimeO), Insecure Use of tx.origin (Tx-Origin), and Unchecked External Calls (UE).

The responses from each model were parsed, and the identified vulnerabilities were recorded in a structured format to enable direct comparison with the ground truth.

D. Formal Verification with ESBMC-Solidity

A separate Python module was implemented to execute the ESBMC-Solidity verifier [1] for each contract. The execution targeted the specific contract and function names, using the ESBMC binary compiled with Solidity frontend support. The vulnerabilities detected by ESBMC-Solidity were mapped using the same classification schema applied to the LLM outputs.

E. Evaluation Metrics

For each model (GPT-4.0, Gemini 2.5, Claude 4.0, Grok 3 Beta, and ESBMC-Solidity), performance was evaluated by comparing the detected vulnerabilities to the ground truth. The following metrics were computed:

- True Positives (TP): Correctly identified vulnerabilities;
- False Positives (FP): Incorrectly identified vulnerabilities;
- False Negatives (FN): Missed vulnerabilities;
- True Negatives (TN): Correctly identified absences;

From these, the following statistical measures were derived:

- Recall (Sensitivity) = TP / (TP + FN)
- Precision (Positive Predictive Value) = TP / (TP + FP)

- False Positive Rate (FPR) = FP / (FP + TN)
- False Negative Rate (FNR) = FN / (FN + TP)
- Overall Accuracy = (TP + TN) / (TP + TN + FP + FN)

These metrics provided a comprehensive quantitative basis to compare the models' efficacy in detecting vulnerabilities in smart contracts under practical and reproducible conditions. Methodology summary shown in fig 1.

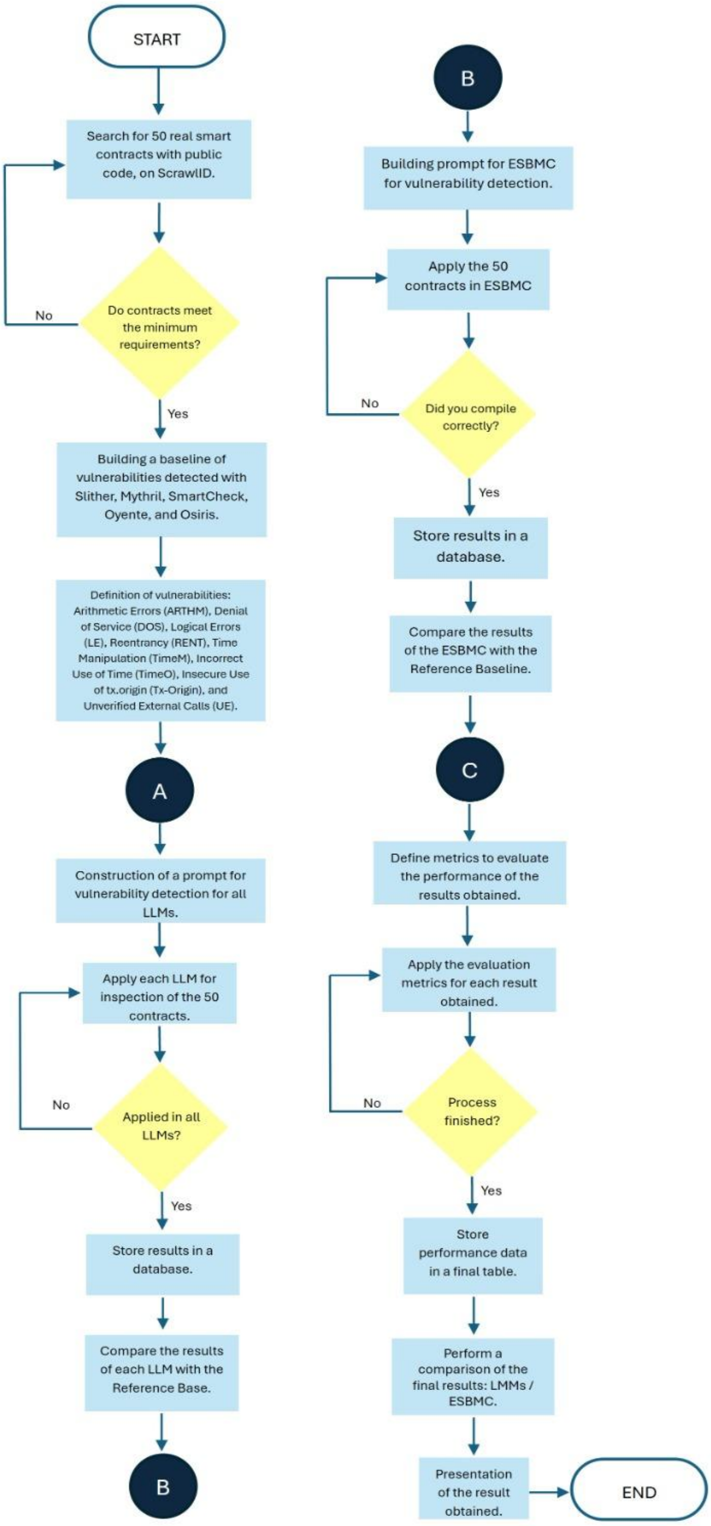
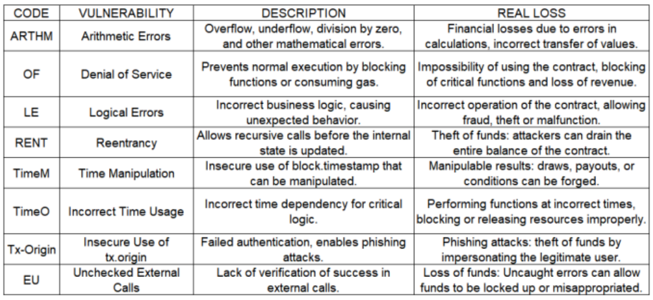

START
Search for 50 real smart contracts with public code, on ScrawlID.
No
Do contracts meet the minimum requirements?
Yes
Building a baseline of vulnerabilities detected with Slither, Mythril, SmartCheck, Oyente, and Osiris.
Definition of vulnerabilities: Arithmetic Errors (ARTHM), Denial of Service (DOS), Logical Errors (LE), Reentrancy (RENT), Time Manipulation (TimeM), Incorrect Use of Time (TimeO), Insecure Use of tx.origin (Tx-Origin), and Unverified External Calls (UE).
A
Construction of a prompt for vulnerability detection for all LLMs.
Apply each LLM for inspection of the 50 contracts.
No
Applied in all LLMs?
Yes
Store results in a database.
Compare the results of each LLM with the Reference Base.
B
B
Building prompt for ESBMC for vulnerability detection.
Apply the 50 contracts in ESBMC
No
Did you compile correctly?
Yes
Store results in a database.
Compare the results of the ESBMC with the Reference Baseline.
C
Define metrics to evaluate the performance of the results obtained.
Apply the evaluation metrics for each result obtained.
No
Process finished?
Yes
Store performance data in a final table.
Perform a comparison of the final results: LMMs / ESBMC.
Presentation of the result obtained.
END

Fig. 1. Methodology flowchart. (*author*)

For this work, the following main vulnerabilities in Smart contracts were analyzed as shown in table 1 below:

TABLE I. TABLE VULNERABILITIES



CODE	VULNERABILITY	DESCRIPTION	REAL LOSS
ARTHM	Arithmetic Errors	Overflow, underflow, division by zero, and other mathematical errors.	Financial losses due to errors in calculations, incorrect transfer of values.
OF	Denial of Service	Prevents normal execution by blocking functions or consuming gas.	Impossibility of using the contract, blocking of critical functions and loss of revenue.
LE	Logical Errors	Incorrect business logic, causing unexpected behavior.	Incorrect operation of the contract, allowing fraud, theft or malfunction.
RENT	Reentrancy	Allows recursive calls before the internal state is updated.	Theft of funds: attackers can drain the entire balance of the contract.
TimeM	Time Manipulation	Insecure use of block.timestamp that can be manipulated.	Manipulable results: draws, payouts, or conditions can be forged.
TimeO	Incorrect Time Usage	Incorrect time dependency for critical logic.	Performing functions at incorrect times, blocking or releasing resources improperly.
Tx-Origin	Insecure Use of tx.origin	Failed authentication, enables phishing attacks.	Phishing attacks: theft of funds by impersonating the legitimate user.
EU	Unchecked External Calls	Lack of verification of success in external calls.	Loss of funds: Uncaught errors can allow funds to be locked up or misappropriated.

IX. RESULTS

The evaluation of GPT-4.0, Gemini 2.5, Claude 4.0, Grok 3 Beta, and ESBMC-Solidity 7.9 reveals significant differences in their ability to identify vulnerabilities in Ethereum smart contracts. Based on standardized metrics including recall, precision, false positive rate (FPR), false negative rate (FNR), and overall accuracy. This section highlights the strengths, limitations, and ideal application contexts for each model.

A. GPT-4.0

GPT-4.0 demonstrated a balanced performance between precision and accuracy, achieving a precision of 43.1%, recall of 21.6%, and overall accuracy of 69%. Its relatively low false positive rate (11.6%) makes it a reliable tool in contexts where alert quality is critical. However, its high false negative rate (78.4%) suggests the model may miss a substantial number of actual vulnerabilities, especially those embedded in complex business logic.

Strength: Low false positives and solid overall accuracy.

Limitation: Limited recall, missing a significant portion of vulnerabilities.

Ideal context: Preliminary assessments or triage phases where minimizing noise is a priority.

B. Gemini 2.5

Gemini 2.5 achieved the highest recall (29.3%) among all LLMs, while maintaining a moderate precision of 49.3% and accuracy of 70.75%. However, the FPR (12.3%) is slightly higher than GPT-4.0’s.

Strength: Best vulnerability coverage among LLMs.

Limitation: Slightly higher false positives may require additional manual review.

Ideal context: Mid-stage security audits prioritizing issue discovery.

C. Grok 3 Beta

Grok 3 Beta delivered the highest overall accuracy (71%), with a precision of 50% and recall of 25.9%. While it missed fewer vulnerabilities than GPT-4.0, it still fell short of full coverage.

Strength: Excellent precision and top accuracy.

Limitation: Moderate recall indicates potential blind spots.

Ideal context: Final audit stages demanding high-confidence results.

D. Claude 4.0

Claude 4.0 performed the weakest, achieving only 11.2% recall, 38.2% precision, and 69% accuracy. Its very high false negative rate (88.8%) undermines its practical reliability, though it maintains the lowest FPR (7.4%).

Strength: Minimal false positives, suitable for highly conservative validation.

Limitation: Severely limited recall.

Ideal context: Secondary validator in risk-averse environments.

E. ESBMC-Solidity 7.9

ESBMC-Solidity 7.9 demonstrated the best overall accuracy (78.75%) and highest precision (73.1%) among all tools evaluated. With a recall of 42.2% and a low FPR of 6.3%, it confirms its value as a precise and robust formal verification tool.

However, despite these strong metrics, ESBMC failed to detect certain critical vulnerabilities in any of the evaluated contracts. Notably, it did not identify any instance of the reentrancy vulnerability (RENT), which is one of the most severe issues in Ethereum smart contracts. Additionally, it showed no detection capability for Denial of Service (DOS), Logical Errors (LE), Time Manipulation (TimeM), tx.origin misuse (Tx-Origin), and Unchecked External Calls (UE).

Strength: Best precision, accuracy, and very low false positives.

Limitation: Misses several high-severity vulnerabilities, particularly at the business logic and semantic level.

Ideal context: Final-stage deterministic verification, particularly for low-level technical flaws and arithmetic correctness. Graph generated after final results in fig 2.

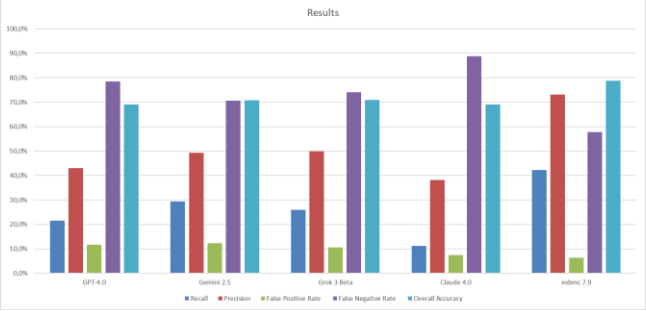


Fig. 2. Final results chart. (*author*)

X. CONCLUSIONS

This study presents a practical and comparative evaluation of four state-of-the-art Large Language Models (LLMs)—GPT-4.0, Gemini 2.5, Claude 4.0, and Grok 3 Beta—against the formal verifier ESBMC-Solidity 7.9, focusing on their ability to detect vulnerabilities in real-world Ethereum smart contracts.

Among LLMs, Grok 3 Beta achieved the best overall accuracy (71%) and precision (50%), while Gemini 2.5 reached the highest recall (29.3%), showing greater coverage of potential issues. GPT-4.0 balanced accuracy and low noise, and Claude 4.0 stood out for its minimal false positives, although it struggled significantly in coverage.

The ESBMC-Solidity 7.9 verifier outperformed all models in overall accuracy (78.75%) and precision (73.1%), with a remarkably low false positive rate (6.3%). These results confirm the high reliability of formal methods for technical verification. However, a key limitation emerged: ESBMC-Solidity failed to identify any instance of critical vulnerabilities such as reentrancy, denial of service, business logic flaws, and semantic issues like Time Manipulation or tx.origin misuse highlighting a significant blind spot in real-world security contexts.

These findings suggest that while formal verification tools like ESBMC remain essential for rigorous and deterministic vulnerability detection, they may be insufficient on their own to address the full spectrum of threats. On the other hand, LLMs, although less precise, demonstrate the ability to detect high-level and contextual vulnerabilities often missed by traditional tools. We compiled the final results[1] to compare the performance of both tools, shown in table 2 below:

TABLE II. TABLE RESULTS

Model	Recall	Precision	False Positive Rate	False Negative Rate	Overall Accuracy
GPT-4.0	21,6%	43,1%	11,6%	78,4%	69,0%
Gemini 2.5	29,3%	49,3%	12,3%	70,7%	70,8%
Grok 3 Beta	25,9%	50,0%	10,6%	74,1%	71,0%
Claude 4.0	11,2%	38,2%	7,4%	88,8%	69,0%
esbmc 7.9	42,2%	73,1%	6,3%	57,8%	78,8%

A hybrid approach that combines the deterministic precision of formal verification with the adaptive reasoning of LLMs presents a promising path forward. LLMs can assist in exploratory analysis and logic flaw detection, while tools like ESBMC can confirm the presence of well-defined technical bugs with mathematical certainty. This complementarity enhances the robustness of smart contract audits and contributes to a safer and more resilient blockchain ecosystem.

XI. OPEN CHALLENGES

Despite notable advances, the verification of smart contracts and critical systems still faces significant open challenges. Formal verification tools like ESBMC require precise configurations, high technical expertise, and are often limited to detecting syntactic or arithmetic errors, with low coverage of high-level logic vulnerabilities. On the other hand, LLM-based approaches introduce a new paradigm of semantic reasoning, but their behavior remains non-deterministic, sensitive to prompt engineering, and difficult to validate formally.

This duality exposes broader challenges in the field of software and systems verification, such as:

[1] https://encurtador.com.br/1H65X

Semantic Understanding at Scale: Bridging the gap between syntactic correctness and contextual understanding of business logic remains a key research frontier, particularly for mission-critical smart contracts.

Explainability and Trust: LLMs often operate as black boxes. Creating explainable and auditable verification outputs is essential to gain trust in automated analyses.

Hybrid Verification Frameworks: Integrating formal tools and AI models into cohesive pipelines presents architectural, computational, and methodological challenges.

Generalization and Benchmarking: The field lacks widely adopted, standardized datasets and evaluation protocols for mixed verification approaches, limiting reproducibility and comparison across studies.

Security Evolution: As new vulnerability types emerge, verification tools must adapt rapidly. Static techniques may lag behind novel attack vectors, while LLMs must be continuously updated and monitored for hallucinations or false positives.

In this context, the work presented here contributes to a growing body of evidence that points toward hybrid, adaptive, and semantically rich verification solutions as the most promising direction. Future research must focus on designing tools that combine the rigor of formal methods with the flexibility and language understanding of AI models, moving closer to reliable, scalable, and intelligent software verification.

References

[1] K. Song et al., "ESBMC-Solidity: An SMT-Based Model Checker for Solidity Smart Contracts," ICSE, 2022.

[2] P. Ince et al., "Generative Large Language Model Usage in Smart Contract Vulnerability Detection," Monash University, 2025.

[3] Y. Sun et al., "GPTScan: Detecting Logic Vulnerabilities in Smart Contracts," NTU Singapore, 2024.

[4] C. S. Yashavant et al., "ScrawlD Dataset," IIT Kanpur, 2022.

[5] P. Ma et al., "Abusing the Ethereum Smart Contract Verification," HUST China, 2023.

[6] W. Ma et al., "Fine-tuning LLM-based Smart Contract Auditing," NTU Singapore, 2024.

[7] S. M. M. Hossain et al., "LLM and ML for Smart Contract Detection," Tennessee Tech, 2025.

[8] M. A. A. Pirzada et al., "LLM-Generated Invariants," Univ. Manchester, 2024.

[9] Z. Xiao et al., "Logic Meets Magic: LLMs for Smart Contract," UNSW Sydney, 2025.

[10] J. Yu, "Retrieval Augmented LLMs," SF State University, 2024.

[11] C. Wu et al., "Semantic Sleuth: Identifying Ponzi Contracts via Large Language Models ," Wuhan University, 2024.

[12] N. Matulevicius, "Verifying Information Flow Security," Univ. Manchester, 2021.