Poster: AGORA: Trust Less and Open More in Verification for Confidential Computing

Hongbo Chen*, Quan Zhou†, Sen Yang‡, Sixuan Dang§,
Xing Han¶, Fan Zhang‡, Danfeng Zhang§, Xiaofeng Wang*
*IU Bloomington, †Penn State, ‡Yale, §Duke, ¶HKUST

***Abstract*—Confidential computing (CC) fails to assure high-level security properties (e.g., no data leakage) on the code. We introduce a novel framework, AGORA, scrupulously designed to provide a trustworthy and open verification platform for CC. To prompt trustworthiness, we observe that specific verification tasks can be delegated to untrusted entities while the corresponding (smaller) validators are securely housed within the trusted computing base (TCB). Moreover, through a novel smart contract-based bounty task manager, it also utilizes crowdsourcing to remove trust in complex theorem provers. These synergistic techniques successfully ameliorate the TCB size burden associated with two procedures: binary analysis and theorem proving. To prompt openness, AGORA supports a versatile assertion language that allows verification of various security policies. Moreover, the design of AGORA enables untrusted parties to participate in any complex processes out of AGORA's TCB.**

I. INTRODUCTION

Confidential computing (CC), powered by hardware Trusted Execution Environments (TEEs), has gained widespread adoption in modern cloud platforms [2]. However, their security primitives fail to address users' evolving demand for security policy compliance. While remote attestation provides cryptographic proof of a TEE program's integrity through hash validation, it fails to establish assurance of code with specific security policies (e.g., software-based fault isolation or SFI). Consequently, users are forced to place blind trust in a hash value rather than being presented with concrete evidence proving the TEE program's security properties.

Prior research has proposed various verification techniques to validate certain security guarantees [1]. However, these solutions partially address the challenge, as they are often tailored to specific scenarios or security properties, resulting in frameworks with reasonable TCB but limited extensibility. Consequently, the verifiability of multiple policies often involves stacking disparate verifiers. Unfortunately, this practice can lead to an inflated TCB, increasing system complexity and potentially introducing new vulnerabilities. Thus, a flexible verification framework is urgently needed to provide auditable proofs to end users and accommodate such scenarios.

General verification frameworks have been developed to verify diverse security policies [5]. However, the implicit trust placed in these heavyweight tools presents significant security risks. Despite extensive testing, the verification stack comprising program analyses and SMT solvers, has exhibited various vulnerabilities. Sophisticated program analyzers (e.g., in eBPF) and SMT solvers have been reported with errors [4], [6] and vulnerabilities (e.g., CVE-2023-2163).

We present AGORA, a verification framework that strikes the best features of the previous two approaches. Compared with general verification frameworks, AGORA is a more open verification service that requires less trust from users. "Open more" implies that an inclusive verification ecosystem welcomes participation from any individual. "Trust less" implies that trust can be removed from the verification ecosystem participants, including the dictatorial parties like SMT solvers. Meanwhile, unlike policy-specific verifiers, the service is compatible with versatile policies and provides a comparable TCB size.

II. SYSTEM DESIGN

A. Overview

Achieving both trustworthiness and openness in one system is challenging: supporting the verification of diverse policies inherently conflicts with trusting less. General verification frameworks often include a series of static analyzers (e.g., alias analysis and dependency analysis), proving techniques (e.g., abstract interpretation and symbolic execution), and SMT solvers as their trusted components, significantly enlarging the TCB. Although generating the results for these analytical tasks can be complex, validating the correctness of their results is much simpler. Thus, our design philosophy is offloading plus validation, delegating complex tasks outside the TCB while maintaining simple validators within the TCB.

AGORA integrates two core components: a *Binary Verifier* (BV) and a *Bounty Task Manager* (BTM), offloading program analysis and constraint solving, respectively. The workflow illustrated in Figure 1 commences with the input binary and assertion. The assertion is synthesized by an untrusted assertion generator conducting program analysis according to the selected policy. The BV then validates the assertions for correctness and formulates verification constraints based on the policy specification. The constraint sets are output to files, which can be verified by constraint solvers. However, as AGORA excludes SMT solvers from the TCB, the BV dispatches the constraint files to the BTM for further inspection. The BTM maintains a bug bounty protocol, accepting submissions of solving results for the constraint sets from bug bounty hunters (BBHs). If a BBH submits a bug to the BTM, the BTM marks the binary as insecure after validation. Finally, the verification results are documented on the blockchain with the certified binary when the crowdsourced solutions are received via the bug bounty protocol. Users can thus access the verification result via blockchain primitives.

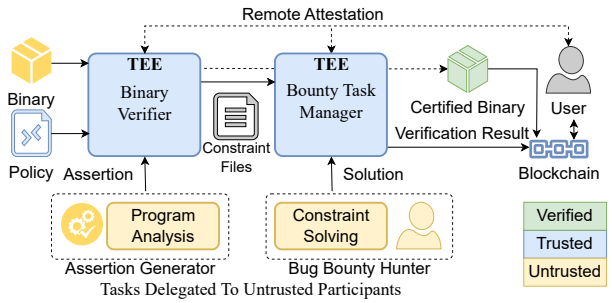


Fig. 1. System architecture and threat model of AGORA verifier.

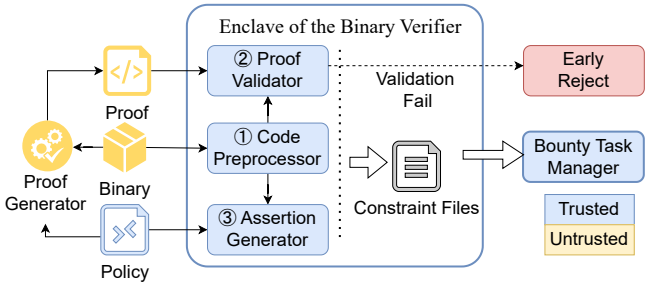


Fig. 2. Overview of the Binary Verifier's workflow.

B. Binary Verification

We highlight several innovative components in AGORA's binary verifier in Figure 2. AGORA supports untrusted *assertion generators* that emit assertions in a meticulously designed language. The language is versatile (i.e., agnostic of verification technique and security policy), striking a balance between expressiveness and simplicity: it can encode various security policies while still allowing a lightweight validator to check the correctness of those untrusted assertions. AGORA further removes the trust in complex theorem provers. The assertions, together with the proof obligations generated for policy compliance checks, form a verification constraint set, which is directed into the BTM for further verification. AGORA integrates SFI policy from VeriWASM [3]. It also supports information flow control and side-channel mitigation policies, which are essential for CC. We demonstrate a concrete example of SFI verification later in the poster.

C. Bounty Task Manager

The BTM solves the constraints from the BV, fostering an inclusive and transparent verification environment with a minimal TCB. It borrows the idea from bug bounty programs and delegates heavyweight theorem proving tasks to untrusted BBHs, validating their solutions via a lightweight protocol. This approach significantly reduces the TCB size. Morever, BBHs can utilize a range of frameworks, such as industry solvers and experimental frameworks, prompting openness.

However, directly adopting traditional bug bounty programs does not work for two reasons: lack of support for no-bug submission (i.e., verification succeeds) and heavy dependency on a trusted third party for bug confirmation and reward distribution. We thus design a new protocol for AGORA.

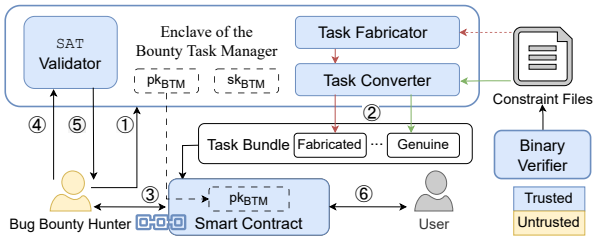


Fig. 3. Workflow of the bug bounty protocol for constraint solving.

Figure 3 shows our protocol. Initially, the BTM receives constraint files from the BV. Upon a BBH's request (①), the *Task Fabricator* and the *Task Converter* collaboratively generate a *task bundle* containing multiple constraint files (②) and publish it via the smart contract. The BBH can then fetch the task bundle from the smart contract (③). After solving a task bundle, the BBH submits their answers for every task (i.e., either `UNSAT` or a `SAT` model) to the BTM (④), which then validates the answers using the *`SAT` Validator*, allocates a reward (⑤), and updates the verification results on the blockchain accordingly. Finally, users can query the verification results (⑥), determining whether to use the binary according to the results.

III. EVALUATION

Our prototype of AGORA consists of 12K lines of source code, achieving more than an order of TCB size reduction compared to solutions with trusted solvers (e.g., `z3` contains ~500K LoC). Also, it achieves equivalent verification power as verifiers implementing identical security policies. For example, benchmarks in SPEC2006 that can be verified by VeriWASM [3] also pass AGORA's check. The protocol discussed in subsection II-C induces a gas cost of $1.18 to verify a binary, which we consider affordable.

REFERENCES

[1] H. Chen, H. H. Chen, M. Sun, K. Li, Z. Chen, and X. Wang, "A verified confidential computing as a service framework for privacy preservation," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 4733–4750.

[2] C. C. Consortium, "Confidential computing: Hardware-based trusted execution for applications and data," Confidential Computing Consortium, Tech. Rep., jan 2021. [Online]. Available: https://confidentialcomputing.io/wp-content/uploads/sites/85/2021/03/confidentialcomputing_outreach_whitepaper-8-5x11-1.pdf

[3] E. Johnson, D. Thien, Y. Alhessi, S. Narayan, F. Brown, S. Lerner, T. McMullen, S. Savage, and D. Stefan, "Доверяй, но проверяй: Sfi safety for native-compiled wasm," in *28th Network and Distributed Systems Security (NDSS) Symposium*, 2021.

[4] J. Park, D. Winterer, C. Zhang, and Z. Su, "Generative type-aware mutation for testing smt solvers," *Proceedings of the ACM on Programming Languages*, vol. 5, no. OOPSLA, pp. 1–19, 2021.

[5] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel *et al.*, "Sok:(state of) the art of war: Offensive techniques in binary analysis," in *37th IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2016, pp. 138–157.

[6] H. Sun and Z. Su, "Validating the {eBPF} verifier via state embedding," in *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, 2024, pp. 615–628.

AGORA: Trust Less and Open More in Verification for Confidential Computing

Hongbo Chen[1], Quan Zhou[2], Sen Yang[3], Sixuan Dang[4], Xing Han[5], Danfeng Zhang[4], Fan Zhang[3], and XiaoFeng Wang[1]
[1]Indiana University, [2]The Pennsylvania State University, [3]Yale University, [4]Duke University, [5]HKUST

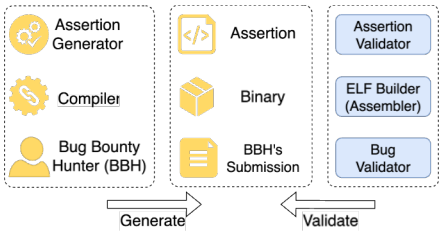
Motivation & Goals

Two problems hinder the adoption of existing verifiers in CC
- Large trusted computing base (TCB) brings potential threats.
- Closed ecosystem rejects public scrutinization and audition.

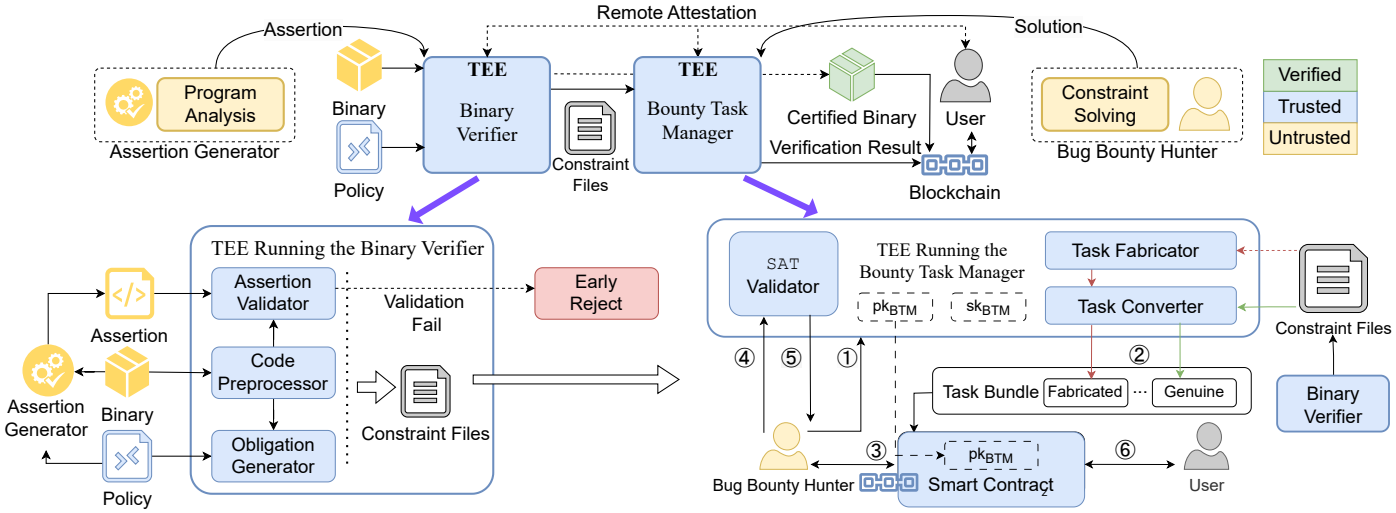
AGORA aims to feature
- Reduced TCB size with dictatorial parties removed.
- Extensible framework supporting versatile security policies.
- Open to untrusted contributions and auditions.

Insight of Design: Offloading

Offloading complex reasoning tasks to untrusted parties, while validating their results using simple and lightweight components within the TCB.



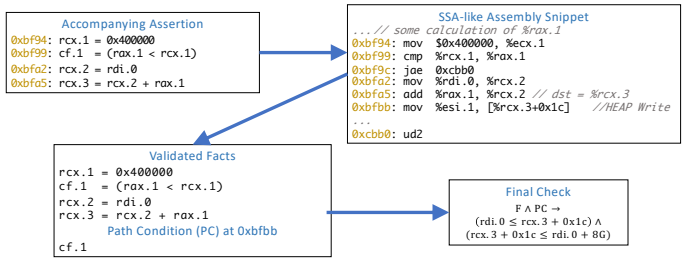
Design Overview



Running Example of Proof Validation: SFI for WebAssembly

SFI Policy for Heap
- All read/write accesses to the 8GB heap memory region must be address-bounded.
- Assumption: at the start of all functions, register **rdi** holds base address of the heap.
- Using static single assignment (SSA) and take **rdi.0** as the initial **rdi**, and **Addr** as the accessed heap address:

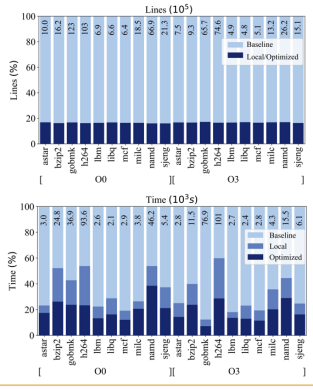
$$\text{HEAP}_{check} \equiv (\text{rdi}.0 \leq \text{Addr}) \wedge (\text{Addr} \leq \text{rdi}.0 + 8\text{G})$$



Evaluation

TCB size comparison (in SLOC) among AGORA, angr, VeriWASM, ConfVERIFY, Deflection, and Cedilla

	General Utilities	Binary Verifier	BTM/ Solver/ Checker	Policy-specific Code	
AGORA	3.6K	848	6.1K	SFI-VeriWASM	625
				IFC-ConfVERIFY	335
				SFI-Deflection	500
				LVI	77
angr	6.6K	67K	532K	HeapHopper	2.1K
				MemSight	3.8K
VeriWASM	2.5K	984	-	SFI	1.9K
ConfVERIFY	~40K	-	-	IFC	1.5K
Deflection	~9.6K	-	-	SFI	~700
Cedilla	~16.5K	4.0K	1.4K	Type-Safety	3.2K



Successfully Verified Policies
- Software-based Fault Isolation (SFI): sandboxing WASM payloads (VeriWASM, Deflection)
- Information Flow Control: tracking flow of sensitive data (ConfLLVM)
- Side-channel Mitigation: targeting load-value injection attack (GNU as assembler)

WASM SFI Verification on SPEC 2006
- Number of lines of constraints reduced by 83%
- Average constraint solving time reduced by 82%