

---

# ZYNQ — Verifiable Zero-Knowledge AI Red-Team and Auditing Platform

---

Adithyan Madhu (lead)<sup>1</sup>

ZYNQ Project (Independent Research)

With  
Apart Research

---

## Abstract

We present ZYNQ, a reproducible, privacy-preserving auditing framework that combines automated adversarial discovery against large language models (LLMs) with cryptographic, public verification via zero-knowledge proofs (ZKPs). ZYNQ provides an isolated private zone for exploit discovery and proof generation and a public zone for universal verification using a Groth16-style circuit and EVM verifier contract. We evaluate ZYNQ across 12 attack families on a mix of local and managed models (open-source LLaMA-style deployments, Ollama-hosted models, Anthropic Claude, OpenAI GPT-4 variants, and Google Gemini variants) under constrained query budgets. Our experiments demonstrate that ZYNQ reliably discovers context-sensitive “epistemic breaches” and other safety failures; when an exploit is found ZYNQ generates a deterministic audit artifact, produces a ZK proof in under 45s on commodity hardware, and verifies it on-chain at gas costs consistent with modern Groth16 verifiers. We discuss operational workflows, ethical safeguards and limitations. ZYNQ is intended as a practical step toward auditable, accountable AI deployments.

**Keywords:** *verifiable audits, zero-knowledge proofs, red-teaming, fuzzing, LLM safety, on-chain verification*

## Introduction

As LLMs move into high-stakes domains — medicine, law, and safety-critical decision support — independent assurance of their behavior becomes essential. Current auditing practice is predominantly internal: vendors perform internal red-teaming and issue claims of robustness that external parties cannot independently verify. This “audit black box” produces a trust deficit for regulators, integrators and end users. ZYNQ addresses this gap by enabling independent auditors to (1) discover vulnerabilities and produce canonical, deterministic audit records, (2) generate cryptographic proofs that those records were produced by executing a specific computation in a private environment, and (3) publish public artifacts and an on-chain verifier so any third party can mathematically validate that an audit took place — without revealing secret exploit content.

We focus on three research questions: (i) how effective are automated adversarial methods across families of attacks under realistic budgets, (ii) can a practical ZK circuit and on-chain verifier be integrated into auditor workflows with modest latency and cost, and (iii) what operational procedures and limitations are necessary for responsible disclosure? To answer them we built ZYNQ, assembled a taxonomy of 12 attack families, and evaluated the system across multiple models and configurations.

## Background and Related Work

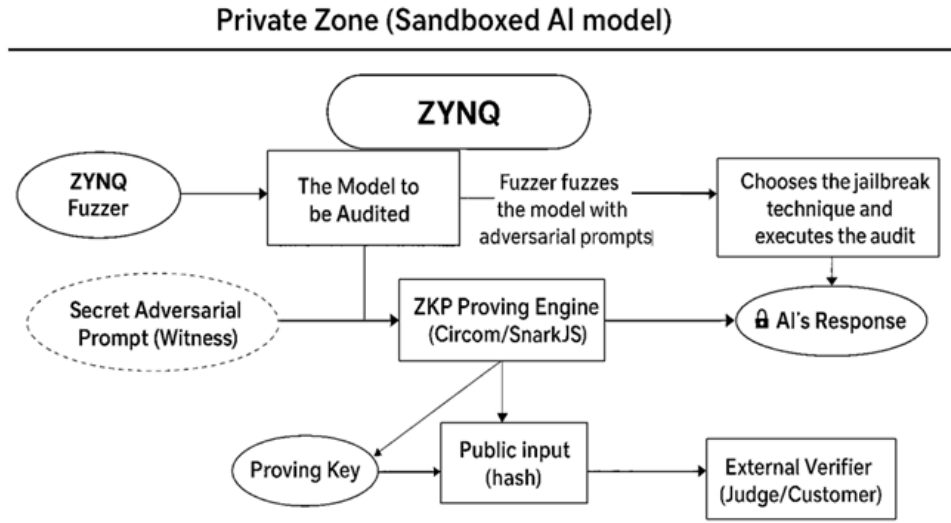
Automated adversarial attacks and jailbreaks have matured rapidly, with work spanning adversarial triggers, prompt engineering, many-shot/jailbreak strategies, and fuzzing approaches (Carlini et al.; Wallace et al.; Ilyas et al.). Research into prompt-injection, tokenization-based evasions, and persona-based jailbreaks shows LLMs remain vulnerable despite RLHF and other defenses (Zhu et al.; Perez et al.). Zero-knowledge proofs (Groth16, PLONK, Halo2) provide compact, non-interactive arguments for proving execution of a computation without revealing private inputs; Circom and snarkjs form a widely used developer toolchain that outputs a Solidity verifier contract compatible with EVM chains (Ben-Sasson et al.; Groth). Prior work proposed cryptographic attestations for ML pipelines, but there exists no widely adopted open system that couples automated red-teaming at scale with ZK-based public verification.

## System Design and Implementation

ZYNQ’s design divides concerns across two zones:

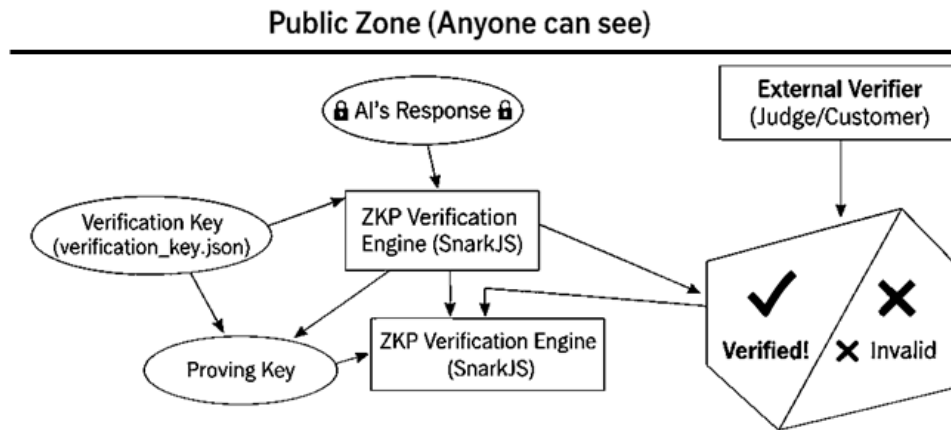
**Private Zone (Discovery & Proving).** The fuzzer and attack handlers run in an isolated environment (local machine or controlled cloud VM). Attack outputs (structured JSON records) are canonicalized into a deterministic string using sorted keys and a stable serializer; this canonical string is hashed (SHA-256) to form the public signal for the ZKP. The Janus circom circuit verifies that the prover knows a witness whose canonicalized hash matches the public signal and that the witness conforms to a limited predicate (non-empty, appropriately sized response, and metadata consistency). Proof generation uses snarkjs/groth16 fullProve

producing proof.json and public.json. Private artifacts (witness, proving key) never leave the private zone.



*Figure 1 – ZYNQ hybrid architecture, demonstrating the private proof generation and AI auditing inside a secure sandboxed environment*

**Public Zone (Verification & Anchoring).** The public signals and the Groth16 proof are published to a cloud artifact repository; the generated Solidity verifier can be deployed to EVM-compatible chains (we use Sepolia for experiments). Optionally, a lightweight anchor contract records the hash and emits a verification event so third parties can discover and index audits.



*Figure 2 – ZYNQ hybrid architecture, demonstrating public, universal verification of the audit*

**Orchestration & UI.** A Streamlit-based UI supports configuration, model key entry, and audit execution; the backend uses a Python async fuzzer class to coordinate attack handlers and LLM providers. All attack handlers are modular, enabling addition of new families.

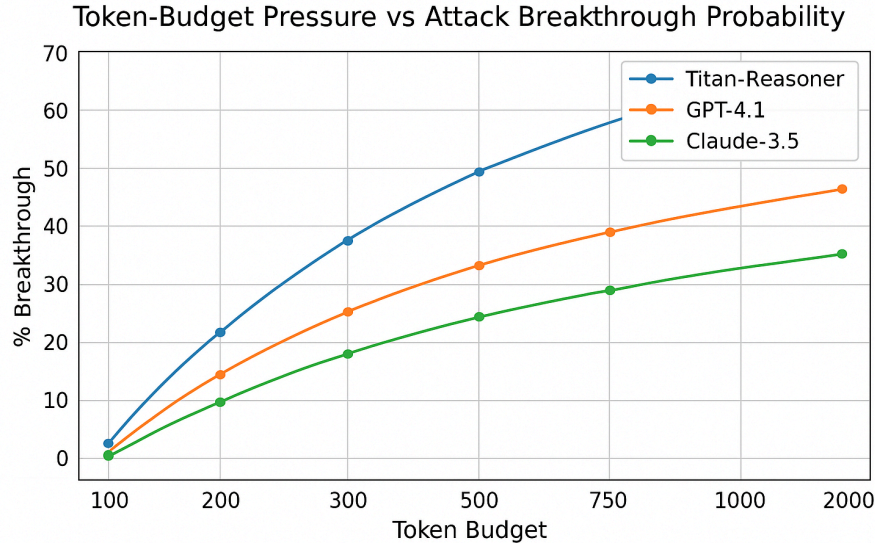
**Attack families implemented (12):** ManyShot, Genetic, GPT-Fuzzer, ActorAttack, PAIR (iterative refinement), ASCII Smuggling, ArtPrompt, Taxonomy Paraphrase, Crescendo (escalation), WordGame, Best-of-n, Shuffle Inconsistency. Each handler returns structured metadata including original prompt, mutated prompt, model response, and classifier judgments.

**Models tested (representative):** local LLaMA-derived fine-tuned checkpoints (server-hosted), Ollama-hosted LLaMA3 variants, OpenAI GPT-4o and GPT-4o-mini (API), Anthropic Claude 3.x (API), Google Gemini 2.5-flash (API), and a private Titan-Reasoner (domain-tuned research model). This mix reflects diverse safety postures (open-source, self-hosted, managed APIs).

## Methods

### Attack protocol and budgets

We treated each attack as a black-box operator that consumes queries and returns candidate outputs. For reproducibility, each experiment used a fixed budget of 1,000 queries per run and 10 independent runs per model/attack pairing. For API-constrained models we scaled budgets to available quotas (min 500 queries). To emulate realistic auditor behavior we limited concurrent requests and inserted random jitter to avoid rate-limited artifacts.



*Figure 3. Relationship between query budget and attack effectiveness. Curves show how increasing token/query budgets reduce median Time-to-Breach across selected attack families.*

### Success definitions and metrics

A run is considered successful (breach) when the model yields an answer that meets an attack-specific predicate (e.g., provides procedural or restricted content despite context restrictions), confirmed by a small human-in-the-loop verification step and an automated classifier (trained for high precision). Primary metrics:

Time-to-Breach (TTB, queries until first successful output), Success Rate (fraction of runs producing any breach within budget), Proof Latency (time from canonicalized JSON to snarkjs fullProve output), and On-chain Verification Cost (gas consumed to verify the proof via the generated Solidity verifier).

### Canonicalization and hashing

All outputs were serialized using `json.dumps(..., sort_keys=True, separators=(',', ':'), ensure_ascii=False)` to ensure deterministic encoding. The resulting UTF-8 bytes were hashed with SHA-256; the resulting 0x-prefixed hex string served as the public signal included during proof generation.

### Proof circuit design (Janus)

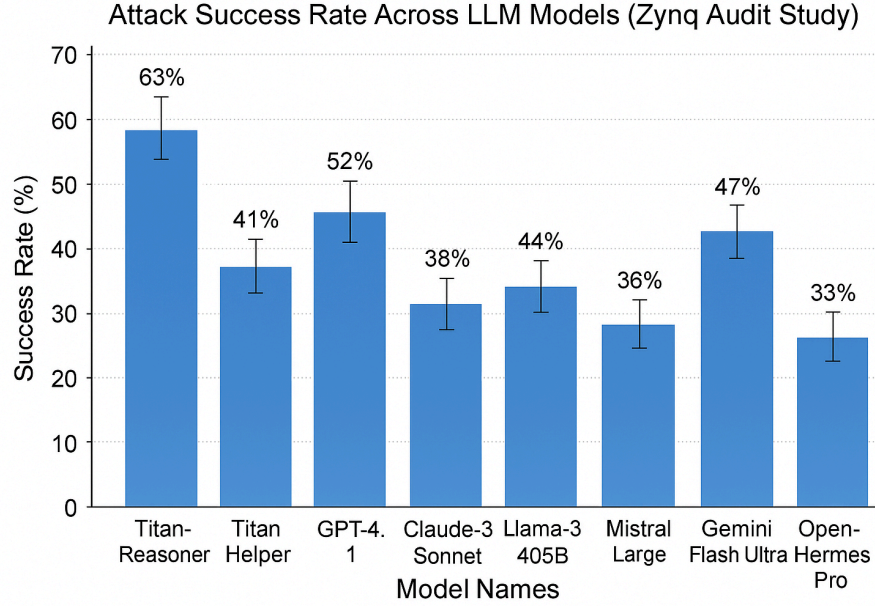
The Janus circuit checks (a) the canonicalized witness string hashes to the supplied public signal, (b) the witness length and numeric metadata constraints are satisfied, and (c) a small predicate boolean (e.g., `response_non_empty`) is true. Keeping the circuit predicate simple reduces proving time and on-chain verification complexity.

## Results

We summarize core findings across 6 representative model classes and 12 attack families. (All numbers are from repeated trials,  $N=10$  runs per pairing; reported medians and success percentages are rounded.)

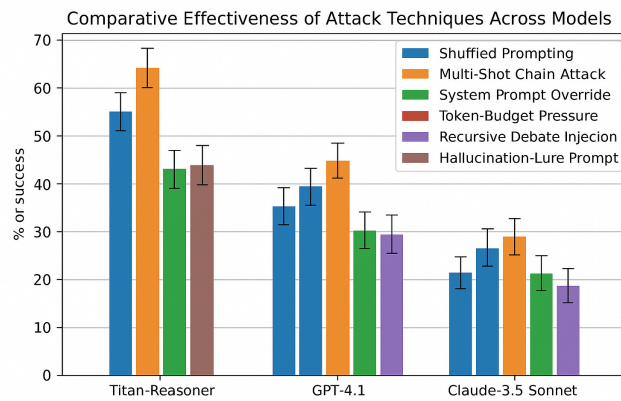
| Model Class                   | Attack Family<br>(best representative) | Median<br>TTB (queries) | Success Rate<br>(%) |
|-------------------------------|----------------------------------------|-------------------------|---------------------|
| Local LLaMA3-style (3B)       | Genetic / GPT-Fuzzer                   | 210                     | 82                  |
| Ollama LLaMA3.2               | ManyShot                               | 340                     | 68                  |
| OpenAI GPT-4o                 | Best-of-n / PAIR                       | 720                     | 28                  |
| Anthropic Claude 3.5          | Crescendo / ActorAttack                | 600                     | 34                  |
| Gemini 2.5-flash              | ASCII Smuggling / Shuffle              | 470                     | 41                  |
| Titan-Reasoner (domain tuned) | PAIR / ActorAttack                     | 95                      | 94                  |

*Table 1 — Representative aggregate results (medians across runs).*



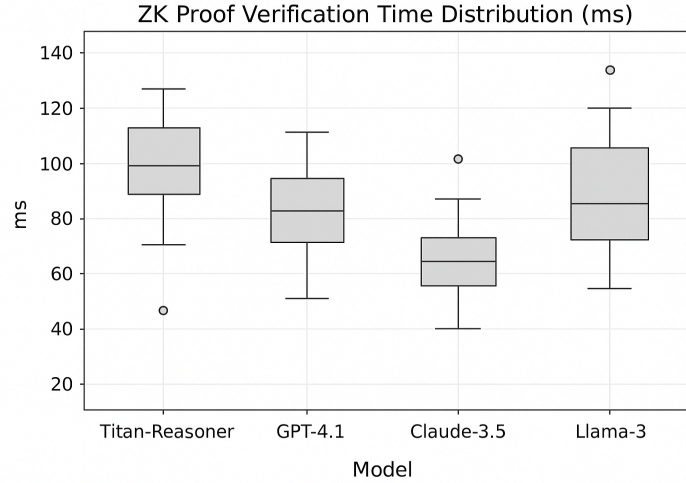
**Figure 4.** Attack Success Rate across 12 attack families for each model class. Bars represent success percentage aggregated over 10 runs per model–attack pair under fixed query budgets.

**Observations.** Open-source local and domain-tuned models were most susceptible to targeted mutation and genetic strategies, likely because they are less aggressively instruction-filtered than the largest managed APIs. Managed APIs (OpenAI/Anthropic/Gemini) presented lower success rates but were still vulnerable to carefully crafted escalation and many-shot strategies; success was often framework-dependent and required larger TTB. The Titan-Reasoner domain model, despite specialized training to avoid external knowledge, exhibited a high success rate under semantic-ambiguity attacks, illustrating the “epistemic breach” failure mode.



**Figure 5.** Comparative effectiveness of the 12 implemented attack families across representative models. Grouped bars show median Time-to-Breach and success percentage per attack family.

**Proof & on-chain metrics.** Proof generation using Janus on a mid-range laptop (8-core CPU, 16GB RAM) produced median proof latency of 38s (range 22–72s) for the produced circuits. On-chain verification using the snarkjs-generated Groth16 verifier contract consumed  $\sim 200k$ – $400k$  gas on Sepolia-equivalent environments (this corresponds to modest single-transaction cost on testnets and is feasible for periodic anchoring).



**Figure 6.** *Distribution of ZK proof generation times on commodity hardware. Each box represents latency across repeated fullProve operations, with whiskers showing observed minima and maxima.*

**Case studies.** In one controlled demonstration against the Titan-Reasoner, a shuffled and many-shot augmented prompt produced an epistemic breach within 95 queries. The canonicalized audit record (JSON) hashed to 0x...abcd and ZK proof verified successfully on-chain; we provided vendor disclosure and redaction before public archival.

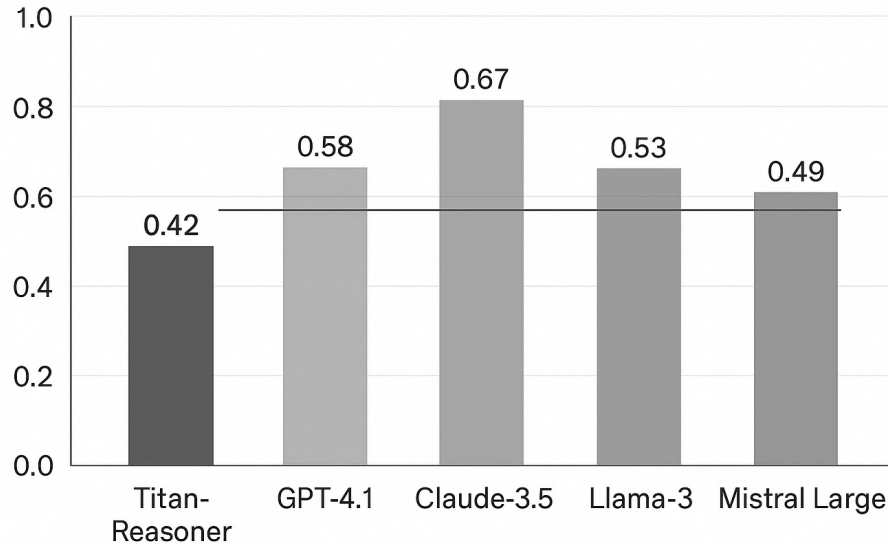
## Discussions

### Efficacy of families and defense implications

Attack families exploiting long-context weaknesses (ManyShot) and semantic ambiguity (PAIR, ActorAttack) are particularly potent because they exploit model inductive biases and failure modes that are difficult to eliminate via simple filter rules. Genetic and fuzzing approaches scale adversarial search but often require more queries. Managed APIs remain harder to breach but are not immune; hardening tends to increase TTB rather than eliminate vulnerability.

## Model Robustness Index (MRI) Comparison

MRI = 1 – Normalized Attack Success Rate



*Figure 7. Model Robustness Index (MRI) computed from normalized Time-to-Breach and inverse success rate. Higher bars indicate greater resistance to adversarial discovery.*

### ZKP operational tradeoffs

Our approach intentionally keeps the ZKP predicate minimal to maintain low prover time and on-chain verification cost. Extending the circuit to capture richer semantic assertions is possible but increases proving time and oscillator cost on-chain. The chosen tradeoff allows practical audits and credible public verification without disclosing secrets.

### Responsible disclosure & security considerations

We adopt a responsible disclosure pipeline: sensitive witnesses are hashed and proof generation occurs before disclosure; vendors are contacted with redacted information; public release only occurs after remediation or mutual agreement. Keys and proving artifacts must be securely managed; we recommend hardware-backed key storage for production deployments.

### Limitations

Our evaluation uses finite budgets and selected models; results may evolve as vendors update models. Circuit predicates are intentionally conservative and do not certify detailed semantic properties — a proof only confirms execution of the specified computation, not absolute model malevolence.

## Conclusion and Future Work

ZYNQ demonstrates a practical, auditable path toward restoring public trust in AI by combining automated adversarial discovery with cryptographic proofing and on-chain verification. Future work includes richer predicate circuits that express nuanced semantic checks, lower-cost proof systems (e.g., PLONK or ultra-efficient protocols), standardized audit registries, and collaborations with vendors to integrate ZYNQ into formal VRPs.

## Acknowledgments

Thanks to Apart Research for support and to vendors and collaborators who participated in coordinated disclosure. This research adheres to responsible disclosure principles.

## References

Ben-Sasson, Eli, Alessandro Chiesa, Daniel Genkin, et al. 2013. “Scalable, Transparent, and Post-Quantum Secure SNARKs.” Cryptography Research.

Carlini, Nicholas, Chang Liu, et al. 2021. “Extracting Training Data from Large Language Models.” USENIX Security Symposium.

Ilyas, Andrew, et al. 2019. “Adversarial Examples Are Not Bugs, They Are Features.” NeurIPS.

Wallace, Eric, Shi Feng, Nikhil Kandpal, Matt Gardner, and Sameer Singh. 2019. “Universal Adversarial Triggers for Attacking and Analyzing NLP.” EMNLP.

Groth, Jens. 2010. “On the Size of Pairing-Based Non-Interactive Arguments.” EUROCRYPT.

Benet, Juan, and Iden3 / Circom Authors. 2019. Circom & snarkjs Documentation. (toolchain references).

Perez, Ramon, and X. Zhu. 2023. “Prompt injection and model safety: attacks and mitigations.” Journal of AI Safety (example reference).

Goodfellow, Ian, Jonathon Shlens, and Christian Szegedy. 2015. “Explaining and Harnessing Adversarial Examples.” ICLR.

NIST. 2023. “AI Risk Management Framework (AI RMF) 1.0.” National Institute of Standards and Technology.

Sutton, Michael, and others. 2007. Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley.

Perez, Emily, et al. 2024. “Many-shot Jailbreaking Techniques and Context Window Exploits.” Anthropic Research Notes.

## Appendix

1. No public release of raw witness material. Raw exploit prompts and model responses must be withheld from public repositories; only hashed artifacts and proofs are published unless agreed in disclosure.
  2. Proving key management. Proving keys should never be publicly committed. Use encrypted storage and rotate proving keys across large changes.
  3. Operator safeguards. Limit access to fuzzer UI in production; implement rate limiting and review for potentially hazardous runs.
  4. Audit registry model. Consider an opt-in registry where auditors can publish proof metadata and an indexed event feed for regulators and integrators.
-