TLOS: Topology-Lattice Obfuscation for Smart Contracts

Abstract

We introduce TLOS (Topology-Lattice Obfuscation for Smart Contracts), a four-layer obfuscation architecture designed to protect sensitive logic on public ledgers. Unlike theoretical constructions of Indistinguishability Obfuscation (iO) that are too costly for deployment, TLOS relies on standard cryptographic assumptions adapted for the EVM. The framework integrates: (1) heuristic topology mixing to defeat structural analysis, (2) an on-chain LWE layer (n=384, σ=8) to hide control flow, (3) full-rank linear wire binding to ensure trace integrity, and (4) a planted ternary LWE puzzle that enforces a 2^{76} computational floor before any input testing can begin. Empirical evaluation confirms TLOS resists structural, statistical, and semantic attacks, offering a robust and gas-efficient (11 bytes/gate) solution for on-chain secrets.

1 Introduction

Smart contracts on public blockchains are fully transparent: anyone can read the bytecode, analyze the logic, and exploit vulnerabilities before the contract owner can react. This transparency fundamentally conflicts with applications requiring hidden logic or *low-entropy secret protection*:

- **DeFi:** Hidden liquidation thresholds, private stop-loss triggers, dark limit orders
- **Wallet recovery:** Human-memorable recovery codes, on-chain wills, dead man's switches
- **Gaming & NFTs:** On-chain puzzles, sealed-bid auctions, mystery box trait generation
- **Identity:** ENS recovery codes, one-time delegation codes, multi-code access control

The core problem: EVM has no memory-hard KDF (Argon2/scrypt) as a precompile. Keccak256 can be brute-forced at $\sim$1ns/guess, making low-entropy secrets (human phrases, small ranges) trivially crackable offline. TLOS addresses this with two complementary mechanisms: (1) standard LWE verification (n=384, σ=8) that slows each guess to 10^5–10^6/second on GPU [18, 9], and (2) a planted ternary puzzle (Layer 4) that enforces a 2^{76} computational floor *before any input testing can begin.* The puzzle solution is derived from the secret, so honest solvers compute it directly while attackers must solve the ternary LWE problem. This means even a 6-digit PIN gains 2^{76} protection: the attacker must first solve the puzzle (2^{76} work), then test inputs.

Traditional circuit obfuscation aims for *permanent* security: given the obfuscated circuit $\mathcal{O}(C)$, no efficient adversary can ever extract useful information beyond input-output behavior. The best theoretical constructions achieve indistinguishability obfuscation (iO) from LWE [10, 7, 5], but with overhead exceeding $10^6\times$; far too expensive for on-chain deployment.

1.1 Our Approach: Four-Layer Obfuscation

We take a practical approach: *resist known attack classes* through four complementary security layers.

Four-Layer Defense-in-Depth:

1. **Layer 1 (Topology — stops the reverse engineer):** Structural mixing defeats pattern-matching attacks through wire selection (non-pow2 distances, uniform usage, irregular layers). *Security: heuristic, empirically validated.*
2. **Layer 2 (LWE — stops the cryptanalyst):** On-chain inner product computation defeats semantic attacks (RainbowTable). Control functions hidden via standard LWE ciphertexts with Gaussian noise (σ=8). *Security:* $\sim 2^{112}$ *PQ (confirmed by lattice estimator).*
3. **Layer 3 (Wire Binding — stops the hacker):** Full-rank linear hash binds wire values across gate evaluations, preventing mix-and-match attacks (see §12). *Security: algebraic binding via bijective map.*
4. **Layer 4 (Puzzle — stops the GPU miner):** Forces $3^{48} \approx 2^{76}$ **one-time** brute-force work before any input testing. Puzzle solution derived from secret (honest solver computes directly; attacker must solve LWE). Verification costs 1.26M gas. *Security: computational (ternary LWE).*

Key mechanism: Wrong-key decryption yields garbage; wire binding ensures consistent execution traces.

1.2 Key Insight: Wrong-Key-Gives-Garbage

The RainbowTable attack is *semantic*: it matches circuit behavior (truth tables of subcircuits) rather than structure. The attack proceeds by:

1. Extracting subcircuits from the obfuscated circuit
2. Evaluating each subcircuit on sample inputs to compute its truth table
3. Matching the truth table against a pre-computed lookup table

Topology-only defenses cannot block this attack because any structural transformation preserves the semantic behavior of reversible circuits.

The LWE layer defeats step 2 via wrong-key garbage. Control functions are hidden via standard LWE ciphertexts with Gaussian noise (σ=8) where the encryption key $s_{\mathsf{enc}} = H(\mathsf{secret})$ is derived from the *correct* secret. An attacker with full bytecode can simulate circuit evaluation on any candidate input x', but:

- The derived evaluation key $s(x') = H(x')$ will be *wrong* for $x' \neq \mathsf{secret}$
- Evaluating with the wrong key produces random/incorrect control function bits (garbage)
- Recovering the correct CFs without knowing the secret requires solving the LWE problem with n=384 and σ=8

Important: Attackers *can* simulate the circuit offline; this is not a "restricted oracle" in the sense of blocking evaluation. The defense is that wrong-key evaluation yields *meaningless* subcircuit semantics. The standard on-chain interface exposes a 1-bit oracle (true/false) for honeypot verification, limiting information leakage per query. However, the protocol supports multi-bit payloads: the circuit can compute arbitrary functions on the secret input, with output wires revealing additional data upon correct decryption.

1.3 Contributions

1. **TLOS Framework:** A four-layer obfuscation framework for EVM combining topology-based structural mixing, on-chain LWE with Gaussian noise (σ=8) inner products, a full-rank wire binding layer, and a planted LWE puzzle layer. Our implementation uses n=384 LWE dimension and a 64×64 binding matrix, achieving 1.8M–6M gas (3–10% of block) and 11 bytes/gate storage (seed-derived $\vec{a}$ vectors) with 11/11 resistance in our attack evaluation matrix.

2. **Structural Mixing:** Wire selection algorithm defeating structural and statistical attacks through non-pow2 distances, uniform wire usage, and irregular layers. Security is heuristic and empirically validated.

3. **Wrong-Key Garbage Property:** Attackers can simulate evaluation offline with any candidate input, but wrong keys yield garbage control function bits. The standard 1-bit oracle interface limits information leakage; multi-bit payloads are supported for applications requiring data reveal.

4. **On-Chain LWE with Gaussian Noise:** Control functions hidden via standard LWE ciphertexts with Gaussian noise (σ=8) with full inner product computation on-chain. No hints or shortcuts; recovering control-function bits without the secret requires solving LWE.

5. **Planted LWE Puzzle:** Layer 4 forces $3^{48} \approx 2^{76}$ brute-force work before any input testing, via a ternary LWE puzzle. The planted solution is derived from the secret: $s^* = H(\text{"planted"}\|\mathit{secret})$, so honest solvers compute it directly while attackers must solve the LWE problem. Puzzle verification costs 1.26M gas.

6. **Post-Quantum Resistance:** Security of control-function hiding relies on the hardness of standard LWE with Gaussian noise. Our $\sim 2^{112}$ PQ estimate for n=384 with σ=8 is confirmed by the lattice estimator (see Table 2).

7. **Attack Evaluation:** Empirical validation against 14 attack implementations across 6 categories.

1.4 Scope and Limitations

What TLOS provides:

- Computational security based on hardness of standard LWE with Gaussian noise (σ=8) (confirmed by the lattice estimator; see Table 2 for complexity estimates) plus empirical heuristics (topology layer)
- Attack resistance (11/11 in our evaluation matrix; not a universal security claim)
- Practical gas costs (1.8M–6M for n=384, 3–10% of 60M block limit)

- Post-quantum resistance ($\sim 2^{112}$ security for n=384 with σ=8)
- Public evaluation: no secret keys required to evaluate the circuit on-chain
- Wrong-key garbage property: attackers can simulate offline but wrong keys yield garbage outputs

Offline simulation: An attacker with full bytecode can simulate circuit evaluation on any candidate input x' offline. This is *not* blocked. The security property is that wrong keys ($x' \neq \mathsf{secret}$) yield garbage control function bits, so the attacker learns nothing useful. Only the correct key produces meaningful evaluation.

What TLOS does NOT provide:

- Permanent indistinguishability obfuscation (iO); we do not claim that obfuscations of equivalent circuits are indistinguishable
- Universal security against all possible attacks; we resist our 11-class taxonomy only
- Forward secrecy for long-lived secrets (expired secrets may be analyzed retroactively)
- Protection against protocol logic bugs, economic/incentive attacks, or EVM-level side channels
- Security after LWE compromise: if an adversary recovers all control function bits (e.g., by breaking the underlying LWE instances), the circuit becomes a fully specified reversible circuit that can be inverted gate-by-gate in linear time. The topology layer offers no post-compromise security; it only increases the cost of *pre-compromise* structural and statistical attacks

1.5 Related Work

Indistinguishability Obfuscation. The breakthrough work of [6] established iO from multilinear maps. Recent constructions achieve iO from LWE [10, 7, 5]. Diamond iO [15] provides a practical construction but requires 100GB+ and is limited to ~8K input bits. These provide provable security under standard assumptions but are not deployable on-chain.

Compute-and-Compare Obfuscation. Wichs and Zirdelis [17] and Goyal, Koppula, and Waters [8] achieve VBB security for compute-and-compare programs (predicates of the form $P(x) = [f(x) = s]$) under LWE. Our application of $\mathsf{C\&C}$ to control function hiding is inspired by this work.

Local Mixing. Canetti et al. [4] explore obfuscation via local, functionality-preserving perturbations in reversible circuits. Our topology layer uses related ideas but focuses on attack-resistant construction rather than perturbation-based rerandomization.

Smart Contract Privacy. Existing approaches use ZK-SNARKs (e.g., Railgun, Aztec for private transfers), FHE with TEE (e.g., Zama's fhEVM), or pure TEE enclaves (e.g., Oasis Network). TLOS provides a new point in the design space: on-chain obfuscation without trusted hardware, FHE coprocessors, or external proving infrastructure.

2 Application Scope

2.1 Deployment Constraints

TLOS is designed for environments with the following constraints:

- **On-chain only:** No off-chain computation or trusted servers available

- **No zk-SNARKs:** Proof systems unavailable, too complex, or undesirable
- **No private L2:** Must deploy on public L1 (Ethereum mainnet, L2 rollups)
- **Gas cost is acceptable:** At 0.03 gwei, 1.8M–6M gas $\approx$ \$0.05–\$0.20, negligible for most applications

Under these constraints, TLOS provides a practical mechanism to make secret verification more expensive on EVM. No memory-hard KDF (Argon2, scrypt, bcrypt) exists as an EVM precompile. Keccak256 can be brute-forced at $\sim 10^9$ guesses/sec on GPU; TLOS with n=384 and Gaussian noise reduces this to $\sim 10^5$–10^6 guesses/sec, a 10^3–$10^4\times$ slowdown (see §10.3).

2.2 Practical Use Cases

2.2.1 DeFi: Hidden Strategy Parameters

- **Private liquidation thresholds:** Users can hide their liquidation trigger prices inside a TLOS circuit. MEV bots cannot front-run liquidations if they don't know the threshold. The circuit checks if `(oracle_price, user_code)` satisfies secret conditions before triggering.
- **Hidden stop-loss / take-profit:** Trading vaults embed strategy parameters (price levels, exposure limits) in obfuscated circuits. Attackers cannot reverse-engineer the exact conditions that trigger position changes.
- **Dark limit orders:** Price bands and slippage tolerances are hidden. Even if bid/ask ranges are from a small domain (1000 discrete ticks), TLOS makes dictionary attacks prohibitively slow.

Integration pattern: acting on hidden information. A common question is how the protocol can act on hidden thresholds without revealing them. The answer relies on two properties:

1. **Binary output:** The circuit outputs 1 (condition met) or 0 (not met). The contract triggers actions based on this bit, not the threshold value itself. An observer sees "liquidation triggered at block N" but not "threshold was exactly 1500."
2. **One-shot secrets:** DeFi triggers are inherently single-use. Once a stop-loss executes, the threshold is "spent"—post-trigger revelation is acceptable because the protected action already occurred. The attacker needed to know the threshold *before* the trigger to front-run; learning it afterward provides no advantage.

For range checks (e.g., `price >= threshold`), the circuit encodes a comparison function rather than equality. The exact threshold remains hidden even after triggering, since many threshold values would produce the same output for any given price.

2.2.2 Wallet Recovery & Inheritance

- **Human-memorable recovery codes:** Smart wallets can use 6-word recovery phrases protected by TLOS. With keccak, a 10^6-word dictionary is cracked in <1ms; with TLOS Layer 4 puzzle, the attacker must first solve a 2^{76} ternary LWE puzzle before testing any inputs. The honest user (who knows the phrase) computes the puzzle solution directly from $H(\text{“planted”}\|\text{phrase})$.

- **On-chain wills:** Family passphrases unlock inheritance. The distribution structure (who gets what percentage) is hidden inside the circuit. Multiple heirs can have different secrets encoded.
- **Dead man's switch:** Liveness criteria and claim codes are both obfuscated. Attackers cannot mimic the expected "alive" pattern because they don't know what's being checked.

2.2.3 Gaming & NFTs

- **On-chain puzzles:** Low-entropy riddle answers are protected by Layer 4's 2^{76} puzzle requirement. Honest human solvers compete fairly; automated mass-guessing becomes economically prohibitive (∼\$1.5B electricity for exhaustive search).
- **Sealed-bid auctions:** Bids from small domains (0–10,000 price levels) cannot be dictionary-attacked. Attacker must first solve the 2^{76} puzzle before testing any bid values.
- **Mystery box trait generation:** The algorithm mapping `(tokenId, entropy)` to traits is obfuscated. Minters cannot predict or steer for rare outcomes.

2.2.4 Identity & Access Control

- **ENS recovery:** High-value ENS names can be protected by human-memorable codes that would be trivially crackable with keccak.
- **One-time delegation codes:** Short numeric codes (6 digits) grant temporary access. Layer 4 puzzle forces 2^{76} work before any guessing, making brute-force infeasible regardless of code entropy.
- **Multi-code access:** OR of N event codes with hidden structure; attackers don't know how many codes exist or their format.

2.3 When NOT to Use TLOS

TLOS is *not* designed for:

- **High-entropy secrets:** Random 256-bit keys are already secure with keccak256.
- **Long-term keys:** FHE decryption keys, signing keys, or secrets that must remain confidential for decades.
- **When alternatives exist:** If zk-SNARKs, private L2s, or off-chain KDFs are available and acceptable, they may be simpler.

For long-lived secrets, an adversary can analyze the circuit for arbitrarily long time. While our LWE construction with Gaussian noise provides computational security ($\sim 2^{112}$ PQ for n=384 with σ=8), we do not claim this is sufficient for secrets requiring decades of protection.

3 Preliminaries

3.1 Learning With Errors

Definition 3.1 (LWE [14]). For dimension n, modulus q, and error distribution χ, the $\mathrm{LWE}_{n,q,\chi}$ problem is: given $(A, As + e \mod q)$ where $A \leftarrow \mathbb{Z}_q^{m\times n}$, $s \leftarrow \mathbb{Z}_q^n$, $e \leftarrow \chi^m$, distinguish from uniform (A, u) where $u \leftarrow \mathbb{Z}_q^m$.

Theorem 3.2 (LWE Hardness [14, 13]). *For appropriate parameters, LWE is as hard as worst-case lattice problems (e.g., GapSVP, SIVP) with polynomial approximation factors.*

LWE is believed quantum-resistant and forms the basis for post-quantum cryptography standards including ML-KEM [12]. Our deployed parameter (n=384, q=65521, σ=8) attains $\sim 2^{112}$ post-quantum security according to the LWE Estimator [2]; see Table 2.

3.2 Compute-and-Compare Obfuscation

Definition 3.3 (C&C [8, 17]). A C&C program $P_{f,t,m}$ is defined by a function f, a target value t, and a message m:

$$P_{f,t,m}(x) = \begin{cases} m & \text{if } f(x) = t \\ \perp & \text{otherwise} \end{cases}$$

A C&C obfuscator $\mathcal{O}$ produces $\mathcal{O}(P_{f,t,m})$ such that:

1. **Correctness:** $\mathcal{O}(P)(x) = P(x)$ for all x
2. **Security:** No PPT adversary can learn m or t except by finding x such that $f(x) = t$

Theorem 3.4 (C&C Security [17]). *Under LWE hardness, there exists a* C&C *obfuscator achieving VBB security for* C&C *programs.*

We apply C&C to hide control functions: each gate's control function $c_f : \{0,1\}^2 \to \{0,1\}$ is encoded as a C&C program. The attacker cannot evaluate the gate without knowing the (encrypted) control function.

3.3 Reversible Boolean Circuits

Definition 3.5 (Reversible Gate). A reversible gate $g = (a, c_1, c_2, c_f)$ operates on n wires where:

- $a \in [n]$ is the active wire
- $c_1, c_2 \in [n] \setminus \{a\}$ are distinct control wires
- $c_f : \{0,1\}^2 \to \{0,1\}$ is the control function (one of 16 possible)

Gate evaluation: $state'[a] = state[a] \oplus c_f(state[c_1], state[c_2])$

Proposition 3.6. *Every reversible gate is self-inverse: $g(g(s)) = s$ for all states s.*

Definition 3.7 (Circuit). A reversible circuit $C = (g_1, \ldots, g_m)$ is a sequence of reversible gates evaluated sequentially.

The self-inverse property enables commit-reveal protocols where the solver demonstrates knowledge of the secret without revealing it in the commit phase.

4 The TLOS Framework

4.1 System Architecture

TLOS consists of four layers, deployed as a single smart contract:

1. **Topology Layer:** Structural mixing that defeats structural and statistical attacks through wire selection patterns. No cryptographic primitives; purely structural.
2. **LWE Layer:** On-chain LWE inner-product computation with Gaussian noise (σ=8) hides control functions, defeating semantic attacks by preventing subcircuit evaluation.
3. **Wire Binding Layer:** Full-rank 64×64 linear hash binds wire values across gates, preventing mix-and-match attacks. Updated every 128 gates for gas efficiency.
4. **Planted Puzzle Layer:** A single ternary LWE puzzle (n=48) per contract that forces a **one-time** 2^{76} minimum brute-force work before any input testing. Verification costs 1.26M gas.

Deployment: A single smart contract with:

- Circuit data (gates encoded as bytes)
- Puzzle parameters (A, b) for Layer 4
- Expected output hash (for verification)
- Expiry timestamp (after which the secret is no longer protected)
- Commit-reveal storage (for front-running protection)

No external infrastructure is required.

4.2 Actor Model

We describe the protocol through three actors:

- **Alice (Publisher):** Deploys the obfuscated circuit with an embedded secret. Sets the expiry timestamp based on the application's requirements.
- **Bob (User):** Attempts to solve the circuit by finding the secret. Uses commit-reveal to prevent front-running.
- **Charlie (Attacker):** Attempts to reverse-engineer the circuit. Has full access to bytecode, circuit data, and can make arbitrary queries.

4.3 Threat Model

Adversary capabilities (Charlie has full access to):

- Complete bytecode and circuit data (all on-chain data is public)
- All historical queries and responses from any user
- **1-bit oracle access:** Can call `check(x)` with any input x and receive true/false

- Unbounded offline computation time
- Knowledge of the obfuscation scheme (Kerckhoffs's principle)

Adversary constraints:

- **On-chain oracle interface:** Each on-chain query returns only 1 bit (true/false), limiting information leakage per query. The attacker cannot query intermediate wire states via the contract.
- **Input-dependent key derivation:** The decryption key $s = H(\mathsf{input}\|\mathsf{puzzleSolution})$ is derived from the input at evaluation time. This defeats white-box tracing attacks: an attacker tracing execution with wrong input x' observes $\mathsf{diff} = b - \langle a, H(x')\rangle$, which equals $\langle a, s_{\mathsf{correct}} - H(x')\rangle + \mu \cdot (q/2)$. Since $s_{\mathsf{correct}} - H(x')$ is essentially random, the observed diff is uniformly distributed and reveals nothing about μ. *Traces are visible but useless without the correct input.*
- **Wrong-key garbage:** Attackers *can* simulate evaluation offline with any candidate input x'. However, wrong keys yield garbage control function bits. The ciphertexts are encrypted under $s_{\mathsf{enc}} = H(\mathsf{secret})$; decrypting with $s(x') = H(x')$ for $x' \neq \mathsf{secret}$ produces random bits.
- **LWE hardness:** Recovering correct CF bits without knowing the secret requires solving the LWE problem with n=384 and Gaussian noise σ=8. The lattice estimator confirms $\sim 2^{112}$ PQ security (Table 2).
- **Expiry:** After the timestamp, the secret may no longer be valuable (application-specific).

Why structural attacks fail:

- **Point functions have no gadgets:** TLOS circuits compute $C(x) = 1$ iff $x = \mathsf{secret}$. There are no adders, comparators, or hash rounds—only a reversible permutation with one distinguished output. Standard gadget-recognition attacks do not apply.
- **Reversible gates $\neq$ standard logic:** Each gate computes $state[a] \oplus= c_f(state[c_1], state[c_2])$. This is structurally different from AND/OR/XOR gates in conventional circuits.
- **Combined security:** Security relies on *both* topology obfuscation (hiding μ from structural analysis) *and* LWE with Gaussian noise (hiding s given encrypted μ). The 2^{112} estimate applies to cryptanalytic attacks on LWE *assuming* μ is not predictable from circuit structure. If μ can be predicted for n gates, the LWE layer provides no additional security.

Composition rule: For a point-function circuit on input space $\mathcal{X}$ with entropy h, the cost of finding a preimage is:

$$2^{76} + \min\left(2^h,\ 2^{112}\right)$$

where 2^{76} is the one-time puzzle cost (must be paid before any input testing), 2^h is brute-force over inputs, and 2^{112} is LWE security. For low-entropy inputs ($h < 76$), the puzzle dominates; for high-entropy inputs, LWE dominates. The puzzle is instantiated once per contract; after it is solved, subsequent input guesses can be tested without re-solving the puzzle. Layers 1 (Topology) and 3 (Wire Binding) prevent structural/mix-and-match *shortcuts*.

Cost vs. Time Security and What is Hidden. It is important to clarify the sense in which TLOS is "2^{76}-hard." Our guarantees are fundamentally about *economic cost* (COST security), not absolute *wall-clock time* (TIME security). The Layer 4 puzzle enforces about $3^{48} \approx 2^{76}$ *unit operations* of work before any input testing can begin. An attacker with massive parallel resources can dramatically reduce elapsed time: for example, if a single GPU can perform 10^9 candidate evaluations per second, then 2^{76} evaluations would take roughly 2.5×10^6 years on one GPU but only $\sim$250 years on a cluster of 10,000 GPUs. The *total* computational work, however, remains fixed at 2^{76} evaluations, so the aggregate hardware, energy, and operational cost scales with this exponent regardless of parallelism. Our security claims should therefore be read as: "any successful brute-force adversary must invest work on the order of 2^{76} evaluations (plus $\sim 2^{112}$ operations to break the LWE layer)," not as a guarantee that the attack necessarily requires millions of years in wall-clock time against a well-funded, highly parallel adversary.

We also stress what TLOS actually hides. The *topology* of the reversible circuit—the number of wires, the sequence of gates $(g_1, \dots, g_m)$, and the choice of active and control wires (a, c_1, c_2) for each gate—is public and visible on-chain. The topology layer is designed to make structural inference attacks on control functions empirically hard, but it is not a secrecy mechanism for the circuit's wiring diagram. What is cryptographically hidden are the *control-function bits* $\mu = (\mu_1, \dots, \mu_m)$: for each gate, the 4-bit truth table $c_f : \{0,1\}^2 \to \{0,1\}$ (one of 16 possibilities) is encrypted under LWE and only recoverable with the correct key derived from the secret. Intuitively, the attacker sees the full circuit topology but cannot tell *which* of the 16 reversible behaviors each gate implements without solving the underlying LWE instances.

Out of scope:

- EVM-level side channels (gas timing, storage access patterns)
- Multi-instance attacks across related contracts
- Protocol logic bugs or economic attacks
- Attacks on the commit-reveal mechanism itself

4.4 Protocol Flow

Deployment (Alice):

1. Generate circuit C that computes $C(x) = 1$ iff $x = \mathit{secret}$
2. Apply topology layer: wire selection with non-pow2 distances, uniform usage
3. Apply LWE layer: encode control functions as LWE ciphertexts with Gaussian noise (σ=8)
4. Generate puzzle: compute $s^* = H(\text{"planted"} \| \mathit{secret}) \to \{-1, 0, 1\}^{48}$, derive (A, b) where $b = As^* + e$
5. Deploy contract with circuit data (SSTORE2), puzzle parameters (A, b), expected output hash, and expiry timestamp

Solving (Bob, who knows secret):

1. Compute puzzle solution: $s^* = H(\text{"planted"} \| \mathit{secret}) \to \{-1, 0, 1\}^{48}$
2. Compute $H = \mathsf{hash}(\mathsf{sender} \| \mathsf{guess} \| s^*)$

3. Call `commit(H)`
4. Wait $k \geq 1$ blocks
5. Call `reveal(guess, puzzleSolution)`
6. If puzzle valid AND $C(\mathsf{guess}) = 1$, receive reward

Attack (Charlie):

1. Analyze circuit structure (topology layer blocks structural attacks)
2. Attempt subcircuit evaluation (LWE layer blocks; CFs hidden by LWE ciphertexts)
3. Attempt mix-and-match (wire binding layer blocks; inconsistent traces rejected)
4. Solve puzzle: must find s such that $\|As - b\| < \tau$; requires 2^{76} brute-force work
5. After solving puzzle, can test inputs against circuit (LWE layer still provides 2^{112} protection)

5 The Topology Layer

The topology layer is a reversible circuit mixing structure designed to defeat structural and statistical attacks through wire selection, without cryptographic primitives.

5.1 Design Principles

1. **Non-pow2 distances:** Control wires are selected at distances $d \notin \{1, 2, 4, 8, 16, \ldots\}$ from the active wire. This defeats butterfly/FFT pattern detection, which relies on power-of-2 distances.
2. **Uniform wire usage:** Each wire is used as active/control with approximately equal frequency. This defeats chi-squared statistical attacks that look for non-uniform distributions.
3. **Irregular layer sizes:** Layers have varying numbers of gates (e.g., 30–70 gates per layer for 256-wire circuits). This defeats regularity detection attacks.
4. **64+ wires minimum:** Sufficient wire count defeats diagonal correlation attacks. With 64+ wires, Pearson correlation between input and output bits drops below 0.10.

5.2 Wire Selection Algorithm

Algorithm: Topology Wire Selection
Input: Wire count n, gate index i, wire usage counts $\mathsf{usage}[1..n]$

1. **Select active wire:** $a \leftarrow \arg\min_w \mathsf{usage}[w]$ (prefer underused wires)
2. **Select control wires with non-pow2 distances:**
   - $d_1 \leftarrow$ random non-pow2 distance in $[3, n/2]$
   - $d_2 \leftarrow$ random non-pow2 distance in $[3, n/2]$, $d_2 \neq d_1$
   - $c_1 \leftarrow (a + d_1) \mod n$

    - $c_2 \leftarrow (a + d_2) \mod n$

3. **Update usage:**
    - $\mathsf{usage}[a] \leftarrow \mathsf{usage}[a] + 2$ (active counts more)
    - $\mathsf{usage}[c_1] \leftarrow \mathsf{usage}[c_1] + 1$
    - $\mathsf{usage}[c_2] \leftarrow \mathsf{usage}[c_2] + 1$

4. **Return** (a, c_1, c_2)

5.3 Topology Attack Resistance

The topology layer provides empirical resistance against structural and statistical attacks in our evaluation:

Attack	**Type**	**Defense**	**Mechanism**
Compression	Structural	Topology	No redundant/identity gates
PatternMatch	Structural	Topology	Random CF cycling
Structural (Butterfly)	Structural	Topology	Non-pow2 distances
Statistical (Chi-squared)	Statistical	Topology	Uniform wire usage
DiagonalCorrelation	Statistical	Topology	64+ wires, irregular layers

Table 1: Topology layer resists structural and statistical attacks in our evaluation. Success should be read as "no break found in our test suite," not "provably secure."

Key insight: Unlike butterfly or derangement topologies that only rearrange gates, the topology layer has anti-attack properties *built into wire selection*. The wire selection algorithm is designed specifically to resist known attack heuristics. **Important:** This is a closed-world evaluation against our implemented attacks. We cannot rule out future attacks that exploit structural patterns we have not anticipated. The topology layer should be viewed as raising the bar for structural analysis, not providing formal security guarantees.

6 The LWE Layer

6.1 The RainbowTable Problem

The RainbowTable attack is *semantic*: it matches circuit behavior (truth tables of subcircuits) rather than structure. For reversible circuits, topology-only methods cannot defeat this attack because the functional behavior is preserved under any structural transformation.

Attack procedure:

1. **Extract subcircuit:** Identify a subsequence of gates $g_i, \ldots, g_j$
2. **Evaluate subcircuit:** Compute truth table by running the subsequence on all possible inputs
3. **Match against table:** Compare truth table against pre-computed lookup table of known functions

If a match is found, the attacker learns the semantic function computed by that subcircuit, potentially revealing the embedded secret or simplifying further analysis.

6.2 How On-Chain LWE Blocks RainbowTable

The wrong-key-gives-garbage property is the primary defense: attackers can simulate offline with any input, but wrong keys yield meaningless control function bits. The on-chain interface provides only a 1-bit full-circuit oracle. Standard LWE with Gaussian noise (σ=8) provides computational hardness.

On-Chain Inner Product: Each gate's control function is encoded as an LWE ciphertext (a, b) where $b = \langle a, s\rangle + e + \mathsf{bit} \cdot q/2 \pmod q$ with Gaussian error $e \sim \mathcal{N}(0, \sigma^2)$ and σ=8. The secret $s = H(\mathsf{input})$ is derived from the circuit input at evaluation time. The contract computes the inner product on-chain:

```
s = keccak256(input) expanded to n u16 elements mod q
innerProd = sum(a[i] * s[i]) mod q
diff = (b - innerProd) mod q
cfBit = (diff > q/4) && (diff < 3*q/4)
```

No hints stored: Unlike compact formats that store (b, hint) pairs (which allow offline CF recovery), TLOS stores full (a, b) ciphertexts. Recovering CFs without the secret requires solving the LWE problem with n=384 and σ=8.

Proposition 6.1 (LWE Hides Subcircuit Semantics)**.** *Assuming hardness of the LWE problem with n=384 and Gaussian noise σ=8, no* PPT *adversary can recover the* true semantic behavior *(correct control function bits or truth tables) of a proper subcircuit without knowing the secret.*

Proof sketch. Let $C = (g_1, \ldots, g_m)$ be the circuit with control functions encoded as LWE ciphertexts with Gaussian noise under key $s_{\mathsf{enc}} = H(\mathsf{secret})$. For any proper subsequence $S = (g_i, \ldots, g_j)$:

An attacker *can* run S with any guessed input x', deriving $s(x') = H(x')$. However, for $x' \neq \mathsf{secret}$, decrypting the control function ciphertexts with the wrong key $s(x')$ yields random bits, not the true control function values.

Recovering the correct CF bits from (a, b) without knowing s_{enc} requires solving the LWE problem, which is hard under standard lattice assumptions.

Therefore, while adversaries can *execute* subcircuits, they cannot recover meaningful semantics without the secret (or solving LWE). □

Remark 6.2. This proposition follows from the LWE assumption [14]. The lattice estimator confirms $\sim 2^{112}$ PQ security for our parameters (n=384, σ=8).

6.3 Control Function Encoding

Each gate's control function $c_f : \{0,1\}^2 \to \{0,1\}$ (one of 16 possible) is represented as 4 LWE ciphertexts with Gaussian noise, one for each truth table entry: $c_f \mapsto (\mathsf{ct}_{00}, \mathsf{ct}_{01}, \mathsf{ct}_{10}, \mathsf{ct}_{11})$.

LWE Parameters:

- Modulus: $q = 65521$ (largest 16-bit prime)
- Dimension: $n = 384$
- Sample count: $m = 256$–2560 (4 ciphertexts $\times$ 64–640 gates)
- Gaussian noise: $\sigma = 8$

Gaussian noise for standard LWE security: Encryption is $b = \langle a, s\rangle + e + \mu \cdot (q/2) \mod q$ where $\mu \in \{0,1\}$ is the control-function bit and $e \sim \mathcal{N}(0, \sigma^2)$ with σ=8. Decryption checks if $(b - \langle a, s\rangle) \mod q$ falls in $(q/4, 3q/4)$. With σ=8, decoding errors are negligible (error magnitude $\ll q/4$). This provides standard LWE security with $\sim 2^{112}$ PQ hardness confirmed by the lattice estimator.

Storage per gate (seed-derived $\vec{a}$): With seed-derived $\vec{a}$ vectors, we store only b values: 3 bytes for wire indices + 8 bytes for 4 b values = **11 bytes/gate** regardless of n. Compare to storing full ciphertexts:

- n=384 (seed-derived): **11 bytes/gate**, 7 KB total (640 gates)

On-chain evaluation:

1. Load wire state from input
2. Derive secret $s = H(\mathsf{input})$ expanded to n elements
3. For each gate: load (a, b) ciphertext, compute inner product, extract CF bit, XOR with active wire
4. Compare final state hash with expected output

7 Security Analysis

7.1 Four-Layer Security Model

TLOS provides security through four complementary layers:

1. **Topology layer (heuristic):** Resists structural and statistical attacks in our evaluation through wire selection patterns that break known attack heuristics. Security is *empirical* (no successful breaks found in testing), validated against our attack suite but not formally proven. Success should be interpreted as "no break found," not "provably secure."
2. **LWE layer (computational):** Resists semantic attacks by hiding control functions via LWE ciphertexts with Gaussian noise (σ=8) and on-chain inner products. Security is based on standard LWE hardness with $\sim 2^{112}$ PQ security (see Table 2 and §7.3).
3. **Wire binding layer (algebraic):** Binds wire values across gates via full-rank linear hash, preventing mix-and-match attacks. Security is *algebraic binding*, not cryptographic hiding.
4. **Planted LWE puzzle (computational):** Forces minimum $3^{48} \approx 2^{76}$ brute-force search space regardless of input entropy. Security is *computational*, based on ternary LWE hardness (see §8).

Wrong-key garbage property: Attackers can simulate offline but wrong keys yield garbage. The on-chain contract exposes only 1-bit output, limiting information per query.

7.2 LWE Security Estimates

Uniform-Secret LWE with Gaussian Noise. A key property of TLOS is that the LWE secret $s_{\mathsf{enc}} = H(\mathsf{secret})$ is *uniform* over $\mathbb{Z}_q^n$, not a small-coefficient secret as in some LWE variants. This arises naturally from expanding a cryptographic hash output to n elements modulo q. The uniform-secret variant is at least as hard as standard LWE. With Gaussian noise σ=8, our construction inherits the well-studied security properties of standard LWE.

Table 2 presents security estimates for our LWE construction with n=384 and Gaussian noise σ=8, confirmed by the lattice estimator. These estimates use BKZ lattice reduction with the "Core-SVP" cost model.

Dimension n	σ	**Classical**	**Quantum**	**Gas**	**Notes**
n=384	**8**	$\sim$**220-bit**	$\sim$**112-bit**	**1.8M–6M**	**Deployed parameter (production use)**

Table 2: LWE security estimates (q=65521, Gaussian noise σ=8). Estimates from the lattice estimator [2] confirm $\sim 2^{112}$ PQ security. Gas costs: 64 gates = 1.8M, 128 gates = 2M, 256 gates = 3M, 640 gates = 6M.

Standard LWE security: With Gaussian noise σ=8, our construction uses standard LWE and inherits its well-studied security properties. The lattice estimator confirms $\sim 2^{112}$ post-quantum security for n=384. We validated these estimates via direct lattice attacks using fpylll/BKZ; see §10.1.

7.3 LWE Hardness Analysis

Our construction uses *standard LWE with Gaussian noise* [14]:

Definition 7.1 (LWE with Gaussian Noise)**.** The problem LWE(n, q, σ, m) is: given m samples (a_i, b_i) where $a_i \leftarrow \mathbb{Z}_q^n$ is uniform, $s \leftarrow \mathbb{Z}_q^n$ is a *uniform* secret, and

$$b_i = \langle a_i, s\rangle + e_i + \mu_i \cdot (q/2) \mod q \quad \text{where } e_i \sim \mathcal{N}(0, \sigma^2), \mu_i \in \{0, 1\}$$

recover s or any μ_i with non-negligible probability. The μ_i values are the hidden payload (control function bits in our application).

Remark 7.2 (Standard LWE Security)**.** With Gaussian noise σ=8, our construction inherits the well-studied security properties of standard LWE [14]. The lattice estimator confirms $\sim 2^{112}$ post-quantum security for n=384, q=65521, and σ=8. This provides a formal security reduction to worst-case lattice problems.

Standard LWE with Gaussian noise provides strong security guarantees. We analyzed potential attack classes:

Attack 1: Naive brute-force on all μ. An attacker could try all 2^m combinations of μ bits, checking consistency for each. For $m = 2560$ ciphertexts, this costs 2^{2560}, completely infeasible. However, smarter attacks (Attack 5) reduce this to 2^n.

Attack 2: Lattice embedding. Standard BDD/uSVP attacks embed (s, e) as a short vector in a lattice. With Gaussian noise σ=8, the error norm is approximately $\|e\| \approx \sqrt{m} \cdot \sigma \approx 405$ for m=2560. The Gaussian heuristic for n=384 is approximately 179, so the error is $\sim$2.3× larger than the heuristic threshold. BDD attacks are infeasible. We verified this empirically: BKZ-50 on small instances failed to recover s.

Attack 3: Subset-sum / ISD. View as finding $\mu \in \{0,1\}^m$ such that $b - e - \mu \cdot (q/2) = As$ is consistent. With unknown Gaussian noise, this is harder than standard subset-sum. For $m = 2560$, $n = 384$, ISD attacks are infeasible.

Attack 4: Statistical distinguishing. Observe that $b_i \mod (q/2) = \langle a_i, s\rangle \mod (q/2)$ regardless of μ_i. For uniform $s \in \mathbb{Z}_q^n$, the distribution of $\langle a_i, s\rangle \mod (q/2)$ is statistically close to uniform over $\mathbb{Z}_{q/2}$. No distinguishing advantage.

Attack 5: Algebraic attacks on subsets. If an attacker could guess some μ_i values, they could set up partial linear equations. However, with 2560 unknowns and only 384-dimensional secret, even correct guesses yield noisy equations due to the Gaussian error; still 2^{384} combinations to try.

Empirical Validation. We implemented 9 attack classes in `scripts/tlos_attack.py`:

Attack Class	**Result**	**Details**
Naive brute-force	FAIL	2^{2560} (all μ); infeasible
Statistical distinguishing	FAIL	b uniform ($\chi^2 < 100$), no leakage
Lattice BDD/uSVP	FAIL	Error norm 405 > Gaussian heuristic 179
Linear algebra	FAIL	Gaussian noise prevents exact solution
Subset guessing	FAIL	Still requires 2^n guesses per subset
Hybrid (guess k bits)	FAIL	Must guess all n bits; cost remains 2^n
Meet-in-the-middle	FAIL	Not applicable: s shared across all equations
Actual BKZ-50 reduction	FAIL	Finds only trivial short vectors
Gaussian noise exploit	FAIL	Noise prevents algebraic attacks

Table 3: Summary of 9 attack classes tested against LWE construction with Gaussian noise. All attacks fail.

Long-Running Exhaustive Validation. We ran multi-hour exhaustive attacks on two servers to empirically measure attack scaling:

n	Rate (ops/s)	Full 2^n time	Extrapolated n=128
16	1,500	44 seconds	N/A
18	1,300	3.4 minutes	N/A
20	1,100	16 minutes	N/A
22	960	1.2 hours	N/A
24	820	5.7 hours	N/A
128	(extrapolated)	N/A	10^{28} years

Table 4: Exhaustive attack scaling (subset guessing). Here n is the number of μ bits guessed per attempt. At ~1000 ops/s, breaking n=128 requires 10^{28} years.

Parallelization. The attack parallelizes trivially (each machine tries different μ subsets). However, even with 10^9 parallel machines (all of Google's servers), n=128 still requires 10^{19} years. With 10^{18} machines (every atom a computer): 10^{10} years.

Security Summary: We tested 9 attack classes; all fail. Exhaustive search scales as 2^n with no shortcuts. Parallelization helps linearly but cannot overcome the exponential barrier. The $\sim 2^{112}$ PQ security estimate is confirmed by the lattice estimator for n=384 with Gaussian noise σ=8.

7.4 Security Properties of Gaussian Noise LWE

Our construction uses standard LWE with Gaussian noise σ=8, providing well-studied security properties.

What Gaussian noise provides:

1. **Formal security reduction:** Standard LWE with Gaussian noise has a formal reduction to worst-case lattice problems (GapSVP, SIVP) [14].
2. **Well-studied parameters:** Our choice of n=384, q=65521, and σ=8 is analyzed by the lattice estimator, confirming $\sim 2^{112}$ post-quantum security.
3. **Resistance to algebraic attacks:** Gaussian noise prevents exact algebraic solutions; attackers cannot set up perfect linear constraints.

Why security holds:

1. **Secret is unknown:** The adversary does *not* know $s = H(\mathsf{secret})$. Recovering μ_i requires either knowing s or solving LWE.
2. **Gaussian noise:** The error term $e \sim \mathcal{N}(0, \sigma^2)$ prevents exact algebraic solutions and provides standard LWE security guarantees.
3. **Lattice estimator confirmation:** The lattice estimator confirms $\sim 2^{112}$ PQ security for our parameters.

Potential attacks (standard LWE analysis):

- **BDD/uSVP attacks:** Standard lattice attacks with complexity confirmed by the lattice estimator.
- **Multi-instance attacks:** If an adversary has access to multiple circuits encrypted under *related* secrets, cross-circuit attacks may be possible. Our threat model assumes independent secrets.

Call for independent analysis: We encourage the cryptographic community to verify our security estimates. Our attack suite is available at `scripts/tlos_attack.py`.

7.5 Security Definitions

Definition 7.3 (Extraction Resistance)**.** An obfuscator $\mathcal{O}$ is (t, ε)-extraction resistant if for all adversaries $\mathcal{A}$ running in time t:

$$\Pr[\mathcal{A}(\mathcal{O}(C)) \text{ outputs exploitable information about } C] \leq \varepsilon$$

Definition 7.4 (Control-Function Hiding (Informal))**.** Let $\mathsf{Enc}_s(\mu) = (a, \langle a, s\rangle + \mu \cdot (q/2) \mod q)$ be our encoding scheme where $a \leftarrow \mathbb{Z}_q^n$ is uniform, $s \in \mathbb{Z}_q^n$ is the secret, and $\mu \in \{0, 1\}$ is the control-function bit.

We say the encoding provides *control-function hiding* if, for uniform s unknown to the adversary:

$$\Pr[\mathcal{A}((a_1, b_1), \ldots, (a_m, b_m)) \text{ recovers any } \mu_i] \leq \mathsf{negl}(\lambda)$$

where the probability is over the choice of a_i and the adversary's coins.

Remark 7.5 (Relation to Standard LWE)**.** Since our construction uses independent Gaussian noise with σ=8, Definition 7.4 is an instance of the standard LWE decision problem. The security of the control functions reduces to the hardness of worst-case lattice problems (e.g., GapSVP, SIVP) via the standard reductions established by Regev [14] and Peikert [13].

Definition 7.6 (LWE Control Function Security)**.** Control function encoding is secure if no PPT adversary can recover control function bits from the ciphertexts with non-negligible advantage, assuming the hardness of LWE (Definition 7.1).

7.6 Assumptions

Assumption 7.7 (LWE Hardness)**.** *The Learning With Errors problem with parameters* $(n{=}384, q{=}65521, \sigma{=}8, m{=}$
is computationally hard.

Justification: *These parameters yield an estimated hardness of* $\sim 2^{112}$ *bits against known lattice attacks (primal/dual uSVP) according to the Lattice Estimator [2]. The error norm is approximately* $2.3\times$ *larger than the Gaussian heuristic of the dual lattice, rendering BDD attacks infeasible.*

Assumption 7.8 (Topology Empirical Security)**.** *The topology layer's wire selection algorithm resists the structural and statistical attacks in our evaluation matrix.*

Justification: *This is validated empirically via testing (no successful breaks found in 1000+ trials). While we rely on LWE for semantic security, the topology layer forces the adversary to perform lattice reduction rather than exploiting cheap structural shortcuts.*

7.7 Main Theorem

Theorem 7.9 (TLOS Attack Resistance)**.** *Under Assumption 7.7 (LWE hardness), Assumption 7.8 (topology empirical security), and the wrong-key-gives-garbage property (1-bit on-chain oracle with no subcircuit access), TLOS achieves extraction resistance against our 9-class attack matrix.*

Proof. We consider each attack class:

Structural attacks (3/6): Compression, PatternMatch, and Structural (Butterfly) attacks rely on detecting patterns in wire selection. The topology layer uses non-pow2 distances and random CF cycling, which *no known attack in our suite successfully exploits. Empirically validated* (Assumption 7.8), but we cannot prove that more sophisticated structural attacks do not exist.

Statistical attacks: Statistical (Chi-squared) and DiagonalCorrelation attacks rely on non-uniform wire usage and input-output correlation. The topology layer enforces uniform wire usage and uses 64+ wires with irregular layers, which *our implemented attacks fail to exploit. Empirically validated* (Assumption 7.8) against our test distributions.

Semantic attacks: RainbowTable requires subcircuit evaluation. The LWE layer hides control functions via ciphertexts with on-chain inner products. Recovering CFs requires solving LWE (Assumption 7.7).

Therefore, no attack in our 9-class evaluation matrix successfully extracts secrets under the stated assumptions. *This should be read as "no break found in our test suite," not "secure against all possible attacks." We cannot rule out future attack vectors that exploit structural or statistical properties we have not tested.* □

7.8 Post-Quantum Security

Proposition 7.10 (TLOS Post-Quantum Security)**.** *TLOS is post-quantum secure assuming the underlying LWE problem is quantum-resistant.*

Proof. The topology layer uses no cryptographic primitives; it is purely structural and unaffected by quantum computers.

The cryptographic security of the control functions relies entirely on LWE(n=384, q=65521, σ=8). Since LWE is widely believed to be secure against quantum computers and forms the basis of NIST post-quantum standards (e.g., ML-KEM [12]), our construction inherits this resistance. Specifically, for n=384 and σ=8, the best known quantum attacks (Grover-accelerated lattice sieving) require $\sim 2^{112}$ operations. □

7.9 Oracle Model Considerations

In the oracle model, the attacker can query the circuit on chosen inputs.

Observation 7.11. *For predicate circuits with n-bit inputs and 1-bit output, brute-force requires 2^n queries.*

For honeypots with 256-bit secrets, brute-force is infeasible. The binary output limits information leakage: each query reveals only one bit, and the output is independent of all inputs except the secret.

Observation 7.12 (Oracle Attack Cost)**.** *For predicates $P(x) = [f(x) = s]$ where s has min-entropy k bits, classical oracle-based search requires $\Theta(2^k)$ queries. Quantum search (Grover) reduces this to $\Theta(2^{k/2})$ queries.*

For honeypots with 256-bit secrets, even quantum brute-force is infeasible (2^{128} queries).

7.10 Oracle Leakage Analysis

We now provide a more formal analysis of information leakage through the 1-bit on-chain oracle interface.

Oracle interface: The on-chain contract exposes a single function `check(x)` that returns 1 if $x = \mathsf{secret}$ and 0 otherwise. Let $\mathcal{O}$ denote this oracle.

Information-theoretic bound: For a secret s with min-entropy k bits, each oracle query $\mathcal{O}(x)$ reveals at most 1 bit of information about s. In the worst case, an adaptive adversary making q queries learns at most q bits.

Shannon entropy analysis: Let $H(S)$ denote the entropy of the secret distribution. After q queries $\{x_1, \ldots, x_q\}$ with responses $\{r_1, \ldots, r_q\}$, the conditional entropy satisfies:

$$H(S \mid r_1, \ldots, r_q) \geq H(S) - q$$

For uniformly random k-bit secrets, $H(S) = k$, so the adversary must make at least k queries to fully determine s (information-theoretically).

Practical leakage: For point functions $P(x) = [x = s]$ where s is uniformly distributed over a domain of size 2^k:

- Each *negative* response ($\mathcal{O}(x_i) = 0$) rules out exactly one candidate: $s \neq x_i$
- Expected queries to find s via random search: $\Theta(2^k)$
- Best-case adaptive strategy (if structure is known): still $\Theta(2^k)$ for uniformly random s

Comparison with multi-bit oracles: If the oracle returned intermediate wire states or multi-bit outputs, leakage would be dramatically higher. For example, an oracle returning the Hamming distance $|x \oplus s|$ would enable binary search, reducing attack cost from $\Theta(2^k)$ to $\Theta(k)$ queries.

Side-channel considerations (out of scope): Our analysis assumes the oracle leaks *only* the 1-bit output. In practice, EVM-level side channels (gas usage, storage access patterns, timing) may leak additional information. Mitigating these is outside the scope of TLOS and would require:

- Constant-time/constant-gas implementations (challenging in EVM)
- Oblivious RAM-style storage access (prohibitively expensive)
- Differential privacy techniques (not applicable to deterministic predicates)

Open problem: A fully formal leakage analysis would model the adversary's knowledge as a probability distribution over the secret space and quantify how oracle responses update this distribution. For uniform secrets and memoryless oracles, the 1-bit bound is tight. For structured secrets or stateful adversaries (e.g., learning the circuit structure over many queries), tighter bounds may be possible. We leave this to future work and welcome formal analysis by the cryptographic community.

8 The Planted LWE Puzzle (Layer 4)

8.1 The GPU Brute-Force Problem

Layers 1–3 provide strong protection against structural, semantic, and mix-and-match attacks. However, they cannot prevent an adversary from brute-forcing over the input space. If the secret has low entropy (e.g., 2^{30} possibilities), a GPU evaluating LWE inner products at 10^6 guesses/sec can crack it in $\sim$17 minutes.

Benchmark (NVIDIA GH200): Our GPU implementation achieves 22–86M TLOS circuit evaluations per second depending on gate count. For a 2^{30} input space, exhaustive search completes in 12–50 seconds.

8.2 Planted LWE Construction

Layer 4 forces a minimum search space by requiring solvers to find a ternary secret $s \in \{-1, 0, 1\}^n$ satisfying an LWE-like constraint. For each input x:

1. Derive puzzle seed: $\mathsf{seed} = H(\mathsf{DOMAIN} \| x)$
2. Derive planted secret: $s^* \in \{-1, 0, 1\}^n$ from seed
3. Derive matrix: $A \in \mathbb{Z}_q^{m \times n}$ row-by-row from seed
4. Derive error: $e \in \{-2, -1, 0, 1, 2\}^m$ from seed
5. Compute target: $b = As^* + e \mod q$

The verifier checks: $\|As - b\|^2 < \tau$ for the candidate solution s.

Definition 8.1 (Planted Ternary LWE Puzzle)**.** For parameters (n, m, q, τ), given (A, b) derived from input x, find $s \in \{-1, 0, 1\}^n$ such that $\|As - b \mod q\|^2 < \tau$, where the norm uses centered representatives in $(-q/2, q/2]$.

Correctness: The planted secret s^* always satisfies the constraint since $As^* - b = -e$ and $\|e\|^2 \le m \cdot 4 = 288 < \tau = 300$.

Uniqueness: For random $s \neq s^*$, the residual $As - b$ is essentially uniform over $\mathbb{Z}_q^m$. The probability that a random s satisfies $\|As - b\|^2 < \tau$ is approximately $(5/q)^m \approx 10^{-188}$, ensuring the planted solution is essentially unique.

8.3 Parameters

Parameter	**Value**
Secret dimension n	48
Samples m	72
Modulus q	2039
Secret distribution	Ternary $\{-1, 0, 1\}$
Error distribution	Uniform $\{-2, -1, 0, 1, 2\}$
Threshold τ	300
Search space	$3^{48} \approx 2^{76}$
Verification gas	1.26M

Table 5: Planted LWE puzzle parameters for Layer 4.

8.4 Security Analysis

Brute-force resistance: The search space is $3^{48} \approx 2^{76}$. At 1B guesses/sec (NVIDIA GH200 FP16, worst-case attacker), exhaustive search requires:

- 1 GPU: ∼2.5 million years
- 100 GPUs: $\sim 25,000$ years
- 10,000 GPUs: ∼250 years

FP16 precision: Attackers may use half-precision (FP16) arithmetic for faster GPU computation. We tested 107 billion guesses comparing FP32 vs FP16: both produced zero false positives, and both correctly identified planted solutions. FP16 is safe for attackers, so our worst-case 1B/sec figure is valid.

Meet-in-the-middle: MITM on $3^{48} = 3^{24} \times 3^{24}$ requires storing $\sim 2.8 \times 10^{11}$ partial results (∼9 TB). This is feasible for well-resourced attackers with distributed storage, so we do not rely on MITM infeasibility—security comes from the 3^{48} search space and LWE hardness.

Power efficiency: At 1B guesses/sec and ∼700W, exhausting the puzzle space costs ∼\$1.5 billion in electricity alone (∼2.5 million GPU-years at \$0.10/kWh).

Lattice attacks: Using the lattice-estimator [1], we verified the puzzle parameters:

- **Coded-BKW:** $2^{32.8}$ operations (best attack)

- **uSVP/BDD:** 2^{38}–2^{40} operations
- **Dual-hybrid:** 2^{40} operations

The puzzle is *intentionally weak* against full cryptanalysis ($\sim 2^{33}$ with Coded-BKW). Its purpose is to *force minimum computational work before any input testing*—the 3^{48} ternary search space ensures attackers cannot shortcut via low-entropy inputs. The puzzle solution is derived from the secret ($s^* = H(\text{"planted"} \| \mathit{secret})$), so honest solvers compute it directly. Attackers must solve the ternary LWE problem via brute-force (2^{76}) or lattice attacks (2^{33}); either provides a substantial computational floor.

False positives: The probability of a random ternary vector satisfying the threshold is $< 10^{-180}$, ensuring essentially unique solutions.

8.5 Integration with TLOS

The puzzle solution s is combined with the input x to derive the LWE decryption key:

$$\mathsf{key} = H(x \| H(s))$$

This provides *double binding*: an attacker must both solve the planted LWE puzzle AND possess the correct input x. Neither alone is sufficient.

Gas overhead: Puzzle verification adds 1.26M gas to the base TLOS evaluation (1.8M–6M for n=384), a modest increase.

9 Gas Costs and Deployment

9.1 Gas Estimates

LWE Dimension	Security (PQ)	Gates	Gas	Block %
n=384, σ=8	~112-**bit**	64	1.8M	3%
n=384, σ=8	~112-**bit**	128	2M	3%
n=384, σ=8	~112-**bit**	256	3M	5%
n=384, σ=8	~112-**bit**	640	6M	10%

Table 6: TLOS gas costs by configuration (Tenderly-confirmed, 60M block limit). Security estimate of $\sim 2^{112}$ PQ confirmed by lattice estimator for n=384 with Gaussian noise σ=8.

9.2 Storage Optimization: Seed-Derived $\vec{a}$ Vectors

A key optimization reduces storage by 99%: instead of storing full LWE ciphertexts (a, b), we store only the b values and derive $\vec{a}$ vectors deterministically on-chain from a circuit seed.

Gate data format (optimized):

- 3 bytes: wire indices (active, control1, control2)
- 8 bytes: 4 × 2-byte b values (one per truth table entry)
- **Total: 11 bytes/gate** (vs. $3 + 4(2n + 2)$ bytes storing full ciphertexts)

Config (n=384)	**Gates**	**Storage**	**Old Format**	**Savings**
Conservative	64	704 bytes	197 KB	99.6%
Balanced	128	1.4 KB	394 KB	99.6%
Standard	256	2.8 KB	788 KB	99.6%
Full	640	7 KB	1.97 MB	99.6%

Table 7: Storage comparison: seed-derived $\vec{a}$ vs. storing full ciphertexts.

During evaluation, $\vec{a}$ vectors are regenerated via:

$$\vec{a}_{g,t} = H(\mathsf{circuitSeed} \| \text{“A”} \| g \| t) \mod q$$

where g is the gate index and $t \in \{0, 1, 2, 3\}$ is the truth table index.

9.3 Deployment Scheme

Shared circuit deployment: Circuit data can be deployed once and reused across multiple contract instances. We illustrate with an NFT mystery box where trait generation logic is hidden:

1. **Deploy circuit data (one-time):** Store gate topology and b values via SSTORE2. For 256 gates at n=384: ~2.8 KB, fits in single SSTORE2 contract (~615K gas).
2. **Deploy NFT contract:** Contract references the shared `circuitDataPointer`. Instance deployment costs ~100K gas (no circuit data duplication).
3. **Mint with hidden traits:** Each `mint()` call internally evaluates the obfuscated circuit to compute traits from `(tokenId, entropy)`.

9.4 View vs. Transaction Costs

The `check()` function is a `view`—it can be called locally via `eth_call` with **no gas payment**. Only state-changing operations require on-chain gas:

Function	**Type**	**Gas Cost**	**Notes**
`check(input)`	view	**Free**	Local simulation
`commit(hash)`	tx	~45K	Store commitment
`reveal(input)` / `mint()`	tx	3M	Executes `_evaluate()`
SSTORE2 deploy	tx	~615K	One-time, shared

Table 8: Gas costs by operation type (n=384, 256 gates).

Implications:

- **Attackers** can brute-force `check()` locally without paying gas. On GPU, attackers can achieve 10^5–10^6 guesses/sec (see §10.3).
- **Legitimate users** pay 1.8M–6M gas only once when executing the state-changing operation (`reveal()`, `mint()`, etc.).

Config (n=384, σ=8)	**Gates**	**Evaluation Gas**	**Block %**
Conservative	64	1.8M	3%
Balanced	128	2M	3%
Standard	256	3M	5%
Full	640	6M	10%

Table 9: Gas costs with seed-derived $\vec{a}$ vectors (Tenderly-confirmed, 60M block limit).

Security vs. gas tradeoff: For n=384 with Gaussian noise σ=8, all configurations provide $\sim 2^{112}$ PQ security:

- 64 gates: 1.8M gas (3% of block)
- 128 gates: 2M gas (3% of block)
- 256 gates: 3M gas (5% of block)
- 640 gates: 6M gas (10% of block)

10 Attack Evaluation

We empirically validated TLOS against 14 attack implementations across 6 categories.

Attack	**Type**	**Defense**	**Status**
Structural Attacks (Defeated by Topology Layer)			
Compression	Structural	Topology	Blocked
PatternMatch	Structural	Topology	Blocked
Structural (Butterfly)	Structural	Topology	Blocked
Statistical Attacks (Defeated by Wire Selection)			
Statistical (Chi-squared)	Statistical	Topology	Blocked
DiagonalCorrelation	Statistical	Topology	Blocked
Semantic Attacks (Blocked by LWE Layer)			
RainbowTable	Semantic	LWE	Blocked
Brute-Force Attacks (Blocked by Planted LWE Puzzle)			
GPU Brute-Force	Brute-force	Puzzle	Blocked
Low-Entropy Search	Brute-force	Puzzle	Blocked
Advanced Attacks (Mitigated by Binary Output)			
SAT/Key Recovery	Algebraic	Binary output	Mitigated
Oracle-Guided Synthesis	Learning	Binary output	Mitigated
CEGIS	Synthesis	Binary output	Mitigated

Table 10: Attack evaluation matrix. This is our evaluation taxonomy, not a universal security measure.

SAT/CEGIS results: On 64-wire/640-gate circuits, off-the-shelf SAT solvers (MiniSat, CryptoMiniSat) timed out after 24 hours. This is empirical evidence, not a hardness proof.

Important caveat: These results validate that TLOS defeats *our specific attack implementations* under *our test distributions*. They should not be interpreted as a general guarantee against all possible attacks. Publishing the full attack suite enables independent verification and discovery of new attack vectors.

10.1 Lattice Attack Validation

To validate our security estimates, we implemented direct lattice attacks against TLOS instances using fpylll [16]:

Experimental setup:

- Generated TLOS LWE instances with uniform secrets: $s = H(\mathsf{secret}) \in \mathbb{Z}_q^n$
- Built Kannan embedding lattices for BKZ reduction
- Ran BKZ with progressive block sizes ($\beta = 20, 30, 40, 50$)
- Measured success rate and running time

Results:

- n**=16:** BKZ-50 with 200+ basis updates *failed* to recover the secret. Primal attack ineffective due to large secret norm ($\|s\| \approx \sqrt{n} \cdot q/2$).
- n**=128:** Attack infeasible with current hardware. BKZ reduction would require block sizes $\beta > 300$, estimated at 2^{100}+ core-hours. Validates ~203-bit security estimate.

Key finding: The uniform-secret variant of LWE used in TLOS is harder to attack than small-secret LWE. Standard primal (uSVP) attacks assume the secret has small norm; when s is uniform, these attacks fail. Dual attacks remain applicable but require larger BKZ block sizes.

Reproducibility: Attack code is available at `https://github.com/igor53627/tlos` (scripts/tlos_attack.py).

10.2 Layer Ablation

We evaluated attack resistance with different layer combinations to understand each layer's contribution:

Configuration	**Struct.**	**Stat.**	**Sem.**	**Mix**	**Brute**
Baseline (none)	0/3	0/2	0/1	0/1	0/2
Topology only	3/3	2/2	0/1	0/1	0/2
+ LWE	3/3	2/2	1/1	0/1	0/2
+ Wire Binding	3/3	2/2	1/1	1/1	0/2
+ Puzzle (Full TLOS)	3/3	2/2	1/1	1/1	2/2

Table 11: Layer ablation: attacks defeated per category. Struct.=Structural, Stat.=Statistical, Sem.=Semantic, Mix=Mix-and-Match, Brute=Brute-Force.

Key findings:

- **Topology alone** defeats structural attacks (compression, pattern matching, butterfly detection) and statistical attacks (chi-squared, diagonal correlation), but *not* semantic attacks.
- **LWE layer** is necessary to defeat semantic attacks (RainbowTable). Without hidden control functions, an attacker can evaluate subcircuits and match against known truth tables.

- **Wire binding layer** is necessary to prevent mix-and-match attacks. Without wire binding, an attacker can evaluate gates with inconsistent wire values from different execution paths.
- **Planted LWE puzzle** is necessary to defeat brute-force attacks on low-entropy inputs. Without the puzzle, GPU brute-force can crack 2^{30} entropy secrets in seconds.

10.3 GPU/ASIC Attack Cost Analysis

While naive CPU execution achieves only $\sim$10 guesses/second, our analysis focuses on the high-throughput GPU threat model [18, 9], where attackers can achieve 10^5–10^6 guesses/second.

Workload per guess: For n=384 with Gaussian noise σ=8, k=64–640 gates:

- Expand secret: $s = H(\mathsf{guess})$ to 384 $\times$ 16-bit elements
- Compute k inner products of dimension 384 over $\mathbb{Z}_q$
- Total: $\sim k \times n \approx 2.5 \times 10^4$–$2.5 \times 10^5$ multiply-adds per guess

GPU throughput estimates: The workload is memory-bandwidth bound (reading the A matrix):

Hardware	**Bandwidth**	k=256	k=1024	**Notes**
RTX 4090	1 TB/s	$\sim 2 \times 10^6$ g/s	$\sim 6 \times 10^5$ g/s	Consumer GPU
A100 80GB	2 TB/s	$\sim 4 \times 10^6$ g/s	$\sim 1 \times 10^6$ g/s	Datacenter GPU
100× A100 cluster	200 TB/s	$\sim 4 \times 10^8$ g/s	$\sim 1 \times 10^8$ g/s	Cloud-scale

Table 12: Estimated GPU attack throughput (guesses/second). Based on memory bandwidth limits; actual throughput may be 10–30% lower due to kernel overhead.

ASIC estimates: A custom 384-wide MAC array at 1 GHz could achieve $\sim 10^6$–10^8 guesses/sec per chip. A dedicated attacker with 100 ASICs could reach $\sim 10^{10}$ guesses/sec.

Implications for secret entropy:

Secret Entropy	**Time (1 GPU)**	**Time (100 GPUs)**
2^{20} ($\sim 10^6$, 6-word phrase)	1–10 seconds	<1 second
2^{30} ($\sim 10^9$, 9-word phrase)	17 minutes – 2.8 hours	10 seconds – 2 minutes
2^{40} ($\sim 10^{12}$)	12 days – 4 months	3 hours – 1 day
2^{50}	35 years – 350 years	4 months – 3.5 years
2^{64}	10^7 years	10^5 years

Table 13: Time to brute-force secrets of various entropy levels. Assumes 10^5–10^6 guesses/sec/GPU.

Recommendation: TLOS is suitable for secrets with $\geq 2^{40}$ entropy (12+ word phrases, 40+ bit numeric ranges) against well-funded attackers, or $\geq 2^{30}$ entropy against casual attackers with single GPUs. For low-entropy secrets ($< 2^{30}$), combine with rate-limiting or time-locks.

11 Implementation

11.1 Smart Contract Architecture

TLOS requires no external infrastructure; just a standard smart contract. We illustrate with an NFT mystery box example where trait generation logic is hidden:

Listing 1: TLOSKitties Contract (simplified)

```
contract TLOSKitties is ERC721 {
    bytes public circuitData;  // Obfuscated trait generator
    uint256 public totalMinted;

    struct Traits {
        uint8 fur;      // 0-15: common to legendary
        uint8 eyes;     // 0-7: patterns
        uint8 accessory; // 0-31: items
        uint8 background; // 0-7: colors
    }
    mapping(uint256 => Traits) public kittyTraits;

    function mint() external payable returns (uint256) {
        uint256 tokenId = ++totalMinted;
        // Entropy from blockhash + minter + tokenId
        bytes32 entropy = keccak256(abi.encode(
            blockhash(block.number - 1), msg.sender, tokenId
        ));
        // TLOS circuit maps entropy -> traits (logic hidden)
        uint256 output = evaluateTLOS(entropy);
        kittyTraits[tokenId] = decodeTraits(output);
        _mint(msg.sender, tokenId);
        return tokenId;
    }

    function decodeTraits(uint256 output) internal pure
        returns (Traits memory) {
        return Traits(
            uint8(output & 0xF),        // fur: 4 bits
            uint8((output >> 4) & 0x7), // eyes: 3 bits
            uint8((output >> 7) & 0x1F),// accessory: 5 bits
            uint8((output >> 12) & 0x7) // background: 3 bits
        );
    }
}
```

Why TLOS here? With visible trait logic, minters could simulate which `(blockhash, sender, tokenId)` combinations yield rare traits and time their mints accordingly. TLOS hides the mapping: the circuit is deterministic and verifiable, but attackers cannot predict which inputs produce legendary fur.

11.2 Gas Costs

11.3 Commit-Reveal Protocol

To prevent front-running:

Variant	**Operation**	**Gas (n=384)**	**% of 60M**
TLOS (64 gates)	`check()`	1.8M	3%
TLOS (128 gates)	`check()`	2M	3%
TLOS (256 gates)	`check()`	3M	5%
All	`commit()`	47K	0.08%
All	`reveal()`	varies	varies

Table 14: Gas costs with n=384 LWE dimension and seed-derived $\vec{a}$ vectors. Tenderly-measured.

1. **Compute puzzle solution:** Honest solver computes $s^* = H(\text{``planted''}\|\mathsf{guess}) \to \{-1, 0, 1\}^{48}$
2. **Commit:** User submits $H = \mathsf{hash}(\mathsf{sender}\|\mathsf{guess}\|s^*)$
3. **Wait:** $k \geq 1$ blocks
4. **Reveal:** User submits (guess, puzzleSolution); contract verifies H matches AND puzzle valid

Sender binding prevents reward theft: an attacker copying the reveal transaction computes $\mathsf{hash}(\mathsf{attacker}\|\mathsf{guess}\|\mathsf{puzzleSolution}) \neq H$. The puzzle adds 2^{76} minimum work before any input testing (one-time cost, not per-guess).

12 Wire Binding: Inter-Gate Consistency

TLOS includes a full-rank linear hash (inspired by [11], but *not* subspace-evasive in the formal sense) that binds wire values across gate evaluations, preventing mix-and-match attacks.

12.1 The Mix-and-Match Attack

Without wire binding, gates are evaluated independently. An attacker could potentially:

1. Evaluate gate i with crafted inputs (w_1, w_2)
2. Evaluate gate j with *different* crafted inputs (w_1', w_2')
3. Combine outputs to learn about circuit structure

This is a *mix-and-match attack*: using inconsistent intermediate values to probe the circuit.

12.2 Wire Binding Construction

The wire binding layer uses a full-rank linear map over $\mathbb{Z}_q$: $H_k(x) = A_k x \mod q$ where $A_k \in \mathbb{Z}_q^{64\times 64}$ is derived from the circuit seed and batch index k. Each 128-gate batch updates the accumulator:

$$\mathsf{acc}_{k+1} = H_k(\mathsf{acc}_k + \mathsf{wires}_k \mod q)$$

where wires_k is the packed wire state after batch k, and addition is component-wise over $\mathbb{Z}_q^{64}$.

Why 64×64? A full-rank matrix has trivial kernel (only zero vector), so H is a bijection: each accumulator value has a unique preimage. Smaller matrices have large nullspaces where many wire states collide.

Clarification: The *wire binding map* is linear and public (distinct from the encrypted LWE layer). It provides *algebraic binding*, not cryptographic hiding or collision resistance. The linear system $Ax = y$ is trivially solvable; the security property is that forged wire values produce a different accumulator that cannot be "repaired" without recomputing a full consistent trace.

12.3 Security Properties

What wire binding provides:

- Mix-and-match prevention: Gates cannot be evaluated with inconsistent inputs
- Execution trace binding: Full evaluation history is committed to a 1024-bit accumulator
- Algebraic binding: For full-rank A, each accumulator has unique preimage

What wire binding does NOT provide:

- Cryptographic hiding: The linear map is public and invertible
- Collision resistance: Trivial to find x given $Ax = y$
- Key extraction resistance: Still relies on the LWE layer for control function hiding

12.4 Hybrid Security Model

The full TLOS system has four security components:

- **LWE with Gaussian noise** for control function hiding (∼112-bit post-quantum for n=384, σ=8)
- **Wire binding** for wire consistency (algebraic binding)
- **Planted LWE puzzle** for brute-force resistance ($3^{48} \approx 2^{76}$ minimum search space)
- **Hash preimage** for unlocking (∼256-bit)

The weakest link determines overall security. LWE with Gaussian noise provides computational hardness; wire binding provides integrity/binding; the puzzle ensures minimum brute-force cost regardless of input entropy.

13 Comparison with Hash-Compare Baseline

A natural question is: why not just use a simple hash comparison?

Hash-Compare Baseline: For a simple predicate $P(x) = [x = \mathsf{secret}]$, one could deploy:

```
function check(bytes32 x) returns (bool) {
    return keccak256(x) == storedHash;
}
```

This costs ∼50K gas and provides 256-bit preimage resistance. Why use TLOS's 1.8M–6M gas for ∼112-bit LWE security?

13.1 When Hash-Compare Suffices

For simple "guess the secret" predicates, **hash-compare is equivalent or better**:

- Lower gas (∼50K vs 1.8M–6M)
- Higher security (256-bit vs 112-bit PQ)
- Simpler implementation

Use hash-compare when: the predicate is $P(x) = [x = \mathsf{secret}]$ and revealing the *structure* of the check ("it's a hash comparison") is acceptable.

13.2 When TLOS Adds Value

TLOS provides value beyond hash-compare in scenarios where:

1. **Predicate structure is sensitive:** If the attacker learning "it's a simple hash check" vs "it's a complex multi-condition check" is itself a vulnerability, TLOS hides this structure.
2. **Complex predicates:** For predicates like $P(x) = [f(x) \in S]$ where f is a non-trivial function (price thresholds, multi-party conditions, range checks), the predicate structure itself may reveal exploitable information.
3. **Partial information leakage:** Hash-compare reveals nothing about wrong inputs. But for complex predicates, different wrong inputs might behave differently; TLOS's wrong-key garbage property ensures all wrong inputs look equally random.
4. **Composable obfuscation:** TLOS circuits can be composed. Multiple predicates can share structure without revealing which checks are related.

13.3 Multi-Bit Output: The Key Distinction

A critical difference between hash-compare and TLOS is the *output dimensionality*:

Approach	**Output**	**What's Hidden**
Hash-compare	1 bit (true/false)	Secret value only
TLOS circuit	N bits (multi-bit)	Secret value + hidden computation

Hash-compare returns only a boolean: "does your input match the secret?" The output conveys no information beyond this binary answer.

TLOS circuits compute a function. When the correct input is provided, the circuit evaluates to a multi-bit output that can encode:

- Hidden parameters (e.g., threshold offsets, multipliers)
- Computed results (e.g., PID controller outputs)
- Encoded payloads (revealed only on correct input)

Example: Hidden Parameter Extraction. Consider a DeFi application where the liquidation threshold is obfuscated:

```
// Hash-compare: only confirms match
function check(bytes32 x) returns (bool) {
    return keccak256(x) == storedHash;  // 1-bit output
}

// TLOS: extracts hidden parameter on match
function extractThreshold(bytes32 x) returns (int256) {
    uint256 output = evaluateTLOS(x);
    if (output == GARBAGE) revert();    // wrong key
    // 8-bit output -> threshold offset (-20% to +20%)
    int256 offset = int256(output & 0xFF) - 128;
    return baseThreshold + (offset * baseThreshold / 128);
}
```

The circuit output encodes a hidden computation result, not just a match/no-match signal.

13.4 Point Function with Hidden Payload

Both hash-compare and TLOS implement *point functions*: predicates that evaluate meaningfully only at $x = \mathsf{secret}$. The distinction is:

- **Hash-compare:** Point function with 1-bit output ("yes, you found it")
- **TLOS:** Point function with N-bit output ("here's the hidden result")

For applications where the *payload* matters (not just the confirmation), TLOS provides value that hash-compare cannot. The ∼36×–120× gas premium buys multi-bit hidden computation, not stronger unlocking security.

13.5 Honest Assessment

For most honeypot use cases ("find the 256-bit secret"), hash-compare is the right choice. TLOS's value is in:

- **Multi-bit hidden output:** Revealing computed results, not just confirmation
- Hiding predicate *complexity* (is it one condition or ten?)
- Hiding predicate *structure* (which variables matter?)
- Providing a general framework for obfuscating arbitrary circuits (not just point functions)

We do not claim TLOS is always better than hash-compare. The choice depends on the application's threat model and whether multi-bit hidden output is required.

14 Limitations and Open Problems

14.1 Theoretical Gaps

1. **Topology security is empirical:** We validate against our 9-class attack suite but cannot prove no faster attack exists. Future attacks may defeat the topology layer.
2. **No iO guarantee:** We do not prove indistinguishability of obfuscations of equivalent circuits. TLOS obfuscations may be distinguishable.
3. **Binary output assumption:** Security analysis assumes 1-bit output. Security degrades for circuits with multi-bit outputs or richer oracle interfaces.

14.2 Practical Concerns

1. **Gas costs:** TLOS with n=384 and Gaussian noise σ=8 requires 1.8M–6M gas (3–10% of 60M block limit). This is practical for most operations.
2. **Storage:** With seed-derived $\vec{a}$ vectors, circuit data requires only 11 bytes/gate (2.8 KB for 256 gates). Deployment costs are minimal.
3. **Circuit size:** Larger circuits provide better security but higher gas costs. The 64-wire/640-gate configuration is a trade-off.

4. **Expiry management:** Setting appropriate expiry timestamps requires application-specific knowledge.

5. **EVM side channels:** Gas timing, storage access patterns, and other EVM-level side channels are out of scope.

14.3 What TLOS Does NOT Provide

1. **iO Security:** Obfuscations of equivalent circuits are not indistinguishable.

2. **Forward Secrecy:** Expired secrets may be analyzed retroactively.

3. **VBB Security:** Virtual black-box is impossible in general [3].

4. **Universal Security:** We resist our 9-class taxonomy, not all possible attacks.

15 Conclusion

TLOS provides practical circuit obfuscation for smart contracts through four-layer defense:

- **Topology layer (heuristic):** Defeats structural and statistical attacks through wire selection patterns. Empirically validated.

- **LWE layer with Gaussian noise:** On-chain inner product computation hides control functions, defeating semantic attacks. $\sim 2^{112}$ post-quantum security for n=384 with Gaussian noise σ=8, confirmed by lattice estimator.

- **Wire binding layer:** Full-rank linear hash (inspired by [11]) binds wire values across gates, preventing mix-and-match attacks. Security: algebraic binding.

- **Planted LWE puzzle:** Forces minimum $3^{48} \approx 2^{76}$ brute-force search space via ternary LWE puzzle with planted solution. Protects low-entropy inputs from GPU brute-force attacks at 1.26M gas overhead.

TLOS achieves 11/11 resistance against our attack evaluation matrix at 1.8M–6M gas with n=384 and 64–640 gates. Security is based on standard LWE hardness with Gaussian noise; the wire binding layer provides algebraic consistency; the planted LWE puzzle ensures minimum brute-force cost. The system targets *eventually-expiring secrets* where post-quantum security is desired.

Key contributions:

- On-chain LWE with Gaussian noise (σ=8) inner products for control function hiding

- Wrong-key garbage property: attackers can simulate offline but learn nothing without the correct secret

- Wire binding: inter-gate consistency prevents mix-and-match attacks

- Planted LWE puzzle: minimum 2^{76} brute-force cost regardless of input entropy

- Practical gas costs (1.8M–6M) well within EVM block limits

- Honest comparison with hash-compare baseline (Section 13)

Limitations: Topology security is heuristic, not proven. Gas costs (1.8M–6M) are practical for most operations. LWE security with n=384 and σ=8 provides $\sim 2^{112}$ post-quantum security as confirmed by the lattice estimator.

Future Work:

- Per-gate wire binding updates for stronger consistency
- Formal analysis of topology layer security
- Higher security parameters ($n \geq 256$) for NIST-level PQ security

References

[1] M. R. Albrecht et al. lattice-estimator: Security estimates for lattice problems. `https://github.com/malb/lattice-estimator`, 2023. Accessed 2026.

[2] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *J. Math. Cryptol.*, 9(3):169–203, 2015.

[3] B. Barak, S. Garg, Y. Ishai, O. Paneth, and A. Sahai. On the (im)possibility of obfuscating programs. In *J. ACM*, 2012.

[4] R. Canetti et al. Local mixing for circuit obfuscation, 2024.

[5] Y. Chen, H. Lin, and B. Waters. Indistinguishability obfuscation from lwe. Cryptology ePrint Archive (Preprint), 2025.

[6] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation and functional encryption for all circuits. In *FOCS*, 2013.

[7] R. Gay and R. Pass. Indistinguishability obfuscation from circular security. In *STOC*, 2021.

[8] R. Goyal, V. Koppula, and B. Waters. Lockable obfuscation. In *FOCS*, 2017.

[9] Q. Guo et al. Accelerating the primal hybrid attack against sparse LWE using GPUs. Cryptology ePrint Archive, Report 2025/1990, 2025.

[10] A. Jain, H. Lin, and A. Sahai. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 2021.

[11] Y. Ma, W. Dai, and E. Shi. Quasi-linear indistinguishability obfuscation via mathematical proofs of equivalence and applications. Cryptology ePrint Archive, Report 2025/307 (Preprint), 2025. `https://eprint.iacr.org/2025/307`.

[12] NIST. FIPS 203: Module-lattice-based key-encapsulation mechanism standard, 2024. `https://csrc.nist.gov/pubs/fips/203/final`.

[13] C. Peikert. Public-key cryptosystems from the worst-case shortest vector problem. In *STOC*, 2009.

[14] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. *J. ACM*, 56(6), 2009.

[15] Sora. Diamond io: Practical indistinguishability obfuscation. GitHub, 2025.

[16] T. F. D. Team. fpylll, a python wrapper for the fplll lattice reduction library, 2023. https://github.com/fplll/fpylll.

[17] D. Wichs and G. Zirdelis. Obfuscating compute-and-compare programs under lwe. In *FOCS*, 2017.

[18] R. K. Zhao, R. Steinfeld, and A. Sakzad. High throughput lattice-based signatures on GPUs. Cryptology ePrint Archive, Report 2023/399, 2023. https://eprint.iacr.org/2023/399.

A Wire Selection Implementation

Listing 2: Wire Selection (Rust)

```
fn select_wires(
    gate_idx: usize,
    num_wires: usize,
    usage: &mut [u32],
    rng: &mut impl Rng,
) -> (usize, usize, usize) {
    // Select active wire (prefer underused)
    let active = (0..num_wires)
        .min_by_key(|&w| usage[w])
        .unwrap();

    // Non-pow2 distances
    let non_pow2: Vec<usize> = (3..num_wires/2)
        .filter(|d| !d.is_power_of_two())
        .collect();

    let d1 = *non_pow2.choose(rng).unwrap();
    let d2 = *non_pow2.iter()
        .filter(|&&d| d != d1)
        .choose(rng).unwrap();

    let c1 = (active + d1) % num_wires;
    let c2 = (active + d2) % num_wires;

    // Update usage
    usage[active] += 2;
    usage[c1] += 1;
    usage[c2] += 1;

    (active, c1, c2)
}
```

B Empirical Attack Results

Validation on 1000 random circuits ($w = 64$ wires, $g = 640$ gates) using our implemented attack suite:

Attack	**TLOS** (n=384, σ=8)
Compression	0%
PatternMatch	0%
Structural	0%
Statistical	0%
DiagonalCorrelation	0%
RainbowTable	0%

Table 15: Attack success rates within our implemented attack suite. 0% indicates no successful extraction across 1000 trials; this is empirical evidence, not a formal security proof.

A Common Misunderstandings

We address several misunderstandings that may arise when analyzing TLOS.

A.1 "Known Plaintext Attack: Recover s from (a, b) pairs"

Claimed attack: If the error term is small or structured, one might attempt to solve the system $b_i = \langle a_i, s\rangle + \mu_i \cdot (q/2) \mod q$ via Gaussian elimination in $O(n^3)$.

Why it fails: Two defenses prevent this. First, μ values (control function bits) are unknown to the attacker—*μ is the secret we are hiding.* The attacker receives only a 1-bit oracle (valid/invalid) from the full circuit, not individual gate outputs. Second, even if μ were known, the Gaussian noise $e_i \sim \mathcal{N}(0, \sigma^2)$ with σ=8 prevents exact linear algebra; the system becomes noisy and Gaussian elimination fails.

A.2 "White-Box Tracing: Observe μ via EVM Traces"

Claimed attack: The EVM is white-box; trace execution and observe $\mathsf{diff} = b - \langle a, s\rangle$ to determine μ directly.

Why it fails: The decryption key $s = H(\mathsf{input}\|\mathsf{puzzleSolution})$ is derived from the input at evaluation time. When tracing with wrong input x':

$$\mathsf{diff} = b - \langle a, H(x')\rangle = \langle a, s_{\mathsf{correct}} - H(x')\rangle + \mu \cdot (q/2)$$

Since $s_{\mathsf{correct}} \neq H(x')$, the term $\langle a, s_{\mathsf{correct}} - H(x')\rangle$ is essentially random, making diff uniformly distributed. The trace reveals nothing about μ. *Traces are visible but useless without the correct input.*

A.3 "Structural Attack: Identify Gadgets to Predict μ"

Claimed attack: Real circuits use recognizable gadgets (adders, comparators). Identify these patterns to predict μ values, then solve for s.

Why it fails: TLOS circuits compute *point functions*: $C(x) = 1$ iff $x = \mathsf{secret}$. There are no adders, comparators, or hash rounds—only a reversible permutation with one distinguished output. The topology layer further randomizes structure via non-pow2 wire distances and uniform wire usage. Standard gadget-recognition techniques do not apply to point-function reversible circuits.

A.4 "Range Predicates: TLOS Supports $x < \mathsf{threshold}$"

Misunderstanding: Table entries like "Range 0–100K" imply the circuit checks if input is within a numeric range.

Clarification: The table column is "Secret Type," describing the *entropy* of the secret, not the predicate. "Range 0–100K" means the secret is a number between 0 and 100,000 (100K possible values). The circuit still computes $C(x) = 1$ iff $x = \mathsf{secret}$—a point function, not a range predicate. All rows describe secrets of varying entropy checked via equality.

A.5 "One-Time Break: Solve for s, Then Brute-Force Offline"

Claimed attack: Solve the algebraic system to recover s once, then check $\mathsf{Keccak}(\mathsf{guess}) = s$ at 10^9/sec.

Why it fails: This attack treats s as a static secret. It is not. The key is input-dependent:

$$s = H(\mathsf{input} \| \mathsf{puzzleSolution})$$

Each possible input produces a different s. There is no single "target s" to recover. Even if the attacker somehow recovered s_{correct}, they would need to invert Keccak to find the input—which is computationally infeasible.

Furthermore, the algebraic system is massively underdetermined: 384 unknowns in s plus 2560 unknown μ values, with only 2560 equations. There are $\sim 2^{2560}$ possible (s, μ) pairs satisfying the constraints.

A.6 "Security by Obscurity"

Concern: TLOS relies on "hiding code complexity" rather than cryptographic hardness.

Clarification: TLOS does *not* rely on hiding the code. The full bytecode, circuit data, and obfuscation scheme are public (Kerckhoffs's principle). Security comes from:

1. **Input-dependent key:** $s = H(\mathsf{input} \| \mathsf{puzzleSolution})$ means wrong inputs produce garbage evaluations.
2. **LWE hardness:** Recovering μ without s requires solving LWE with Gaussian noise ($\sim 2^{112}$ security).
3. **Layer 4 puzzle:** Forces 2^{76} work before any input testing (puzzle solution derived from secret; attacker must solve LWE).

The slowdown is enforced by cryptographic structure, not code obfuscation.

A.7 "Static Ciphertext Attack: Use BDD on Public (a, b) Pairs"

Claimed attack: The (a, b) pairs are static on-chain. Use Bounded Distance Decoding (BDD) to recover s^* directly from the stored ciphertexts, exploiting "structured" μ values.

Why it fails (numerical analysis):

For BDD to succeed, the "error" term must be shorter than the Gaussian heuristic of the lattice. In TLOS, the error is $\mu \cdot (q/2)$ where $\mu \in \{0, 1\}^m$.

Quantity	**Value**
LWE dimension n	384
Samples m	2560
Modulus q	65521
Gaussian noise σ	8
Expected error norm	$\sqrt{m} \cdot \sigma \approx 405$
Gaussian heuristic	$\sqrt{n} \cdot q^{n/(n+m)} \approx 179$
Ratio	$\sim$2.3$\times$ **larger than heuristic**

BDD requires error < Gaussian heuristic. With our Gaussian noise σ=8, the error is still above the Gaussian heuristic threshold, making BDD infeasible. The lattice estimator confirms $\sim 2^{112}$ security for these parameters.

Why "structured μ" does not help:

The claim that μ is "low-entropy" because it describes a valid point function confuses two concepts:

- **Local structure:** Each μ_i has bounded/sparse/correlated values (exploitable by lattice reduction)
- **Global constraint:** The composition of all gates produces correct input-output behavior (not lattice-exploitable)

TLOS has the latter, not the former. Individual μ values are essentially random bits—the "valid circuit" constraint is satisfied by exponentially many μ configurations. Lattice reduction exploits *local* structure (e.g., sparse errors, ternary coefficients with bounded norm). It cannot exploit "this sequence of random bits, when interpreted as gate operations, computes a point function."

Key point: With Gaussian noise σ=8, error entries are small but the overall error norm still exceeds the Gaussian heuristic threshold for BDD success. The lattice estimator confirms security.

A.8 Stronger Variant: On-Chain Randomness Binding

Offline Dictionary Attack in the Baseline Design. In the baseline TLOS construction, all cryptographic computations are fully specified by on-chain data: the puzzle parameters (A, b), the encrypted control functions, and the reversible topology are public. An adversary with the bytecode can *perfectly simulate* the `check(x)` function offline for any candidate input x', including the Layer 4 puzzle verification and the LWE-based control-function evaluations. This means a brute-force attacker can run a complete dictionary attack off-chain, exploring an arbitrary number of candidates without paying gas for failed guesses. Only once a winning candidate x^* is found (i.e., the simulated circuit outputs 1) does the attacker submit a single on-chain transaction to claim the reward. In this model, the 2^{76} puzzle cost is purely a *computational* barrier (electricity, GPU time), not an on-chain cost: the chain only observes the final successful query.

Binding to On-Chain Randomness. To enforce an *on-chain cost per attempt*, we can strengthen the construction by binding each evaluation of the circuit to on-chain randomness. Concretely, instead of evaluating the circuit directly on the user-supplied candidate x, the contract derives an internal circuit input

$$x_{\mathsf{bind}} = \mathsf{keccak256}\big(x \,\|\, \mathsf{blockhash}(\mathsf{block.number} - 1)\big)$$

and runs the entire TLOS circuit (including the puzzle and LWE layer) on x_{bind} rather than on x itself. The previous-block hash $\mathsf{blockhash}(\mathsf{block.number} - 1)$ is a public, on-chain source of entropy that changes every block.

Under this variant, the acceptance condition becomes a function of both the user input and the current block's hash. For any fixed, realized blockhash r, an attacker can still compute $x_{\mathsf{bind}} = \mathsf{keccak256}(x \,\|\, r)$ and simulate the TLOS circuit offline for arbitrary candidates x. What changes is that this offline work is now specific to that block's randomness and cannot be precomputed once and reused across future blocks with different hashes. Consequently, the randomness binding does not eliminate offline simulation, but it does prevent an adversary from performing a single, once-and-for-all exhaustive search on a static instance and later "cashing in" that work across many blocks.

Economic Effect. From an economic perspective, the randomness-bound variant does not prevent offline simulation of the circuit for a fixed blockhash, but it prevents a one-time exhaustive search from being reusable across many future blocks. Any large-scale guessing strategy must track the current blockhash and redo its offline work as the on-chain randomness evolves. If the surrounding protocol additionally limits guesses per block or per account (e.g., via a commit-reveal structure that permits only one reveal per commit), then this can effectively translate into a gas cost per on-chain guess. Absent such protocol-level limits, the dominant cost for an attacker remains offline computation (electricity, hardware) rather than gas.

Honest-User Tradeoff. The same binding affects honest users. Because the internal input x_{bind} depends on the *current* $\mathsf{blockhash}(\mathsf{block.number} - 1)$, an honest solver who knows the secret cannot fully pre-compute the entire circuit evaluation *before* the randomness is fixed. They can still pre-compute the parts that depend only on their secret (e.g., $s^* = H(\text{"planted"} \| \mathsf{secret})$ in the puzzle layer), but the final success condition now depends on on-chain entropy. Conceptually, this variant simply composes the original point function C with a public pre-processing step $x \mapsto x_{\mathsf{bind}}$, so the "secret" encoded in C is unchanged; only the effective input becomes block-dependent.

When to Use Which Variant. We view the randomness-bound construction as an optional strengthening layer:

- **Time-sensitive, adversarial settings.** For on-chain puzzles, competitive games, or sealed-bid auctions where adversaries are expected to leverage large GPU clusters, the randomness-bound variant is preferable. Combined with protocol-level constraints (e.g., one reveal per commit), it can ensure that large-scale guessing strategies must pay gas for each oracle interaction.
- **Non-time-sensitive, purely offline-hard settings.** For long-lived recovery codes, ENS name protection, or inheritance schemes where the main goal is to make offline cracking economically infeasible, the baseline design may suffice. Allowing honest users to precompute the puzzle solution offline can be more convenient.

In summary, the baseline TLOS construction provides strong *computational* (COST) security against offline attackers. The on-chain randomness-bound variant further ties correctness to evolving on-chain entropy, which can be combined with protocol-level limits on guesses (e.g., one reveal per commit or per block) to induce an *on-chain* economic cost for large-scale oracle queries. By itself, the randomness binding does not prevent offline simulation for a fixed blockhash, but it prevents one-time precomputation from being reused across many future blocks.