TLO: Topology-Lattice Obfuscation for Smart Contracts

Abstract

We present TLO (Topology-Lattice Obfuscation), a practical circuit obfuscation framework for smart contracts. TLO uses a two-layer security model: a topology layer that defeats structural and statistical attacks through wire selection patterns (empirically validated), and a lattice layer using on-chain LWE inner products that defeats semantic attacks. Security is based on the computational hardness of the Learning With Errors (LWE) problem combined with heuristic properties of the topology layer.

TLO achieves 6/6 resistance against our attack evaluation matrix at ∼2.58M gas (with n=64 LWE dimension). Control functions are hidden via LWE ciphertexts where the decryption key $s_{\mathsf{enc}} = H(\mathsf{secret})$ is derived from the embedded secret at encryption time; at evaluation time, the evaluator derives $s(x) = H(x)$ from their input x. An attacker can freely simulate evaluation offline with any candidate input x', but wrong keys ($x' \neq \mathsf{secret}$) yield garbage outputs—correct control function bits are recovered only when $s(x) = s_{\mathsf{enc}}$. The on-chain contract exposes only a 1-bit oracle (true/false), limiting information leakage per query. Target applications include predicates with *eventually-expiring* secrets (honeypots, sealed-bid auctions, lotteries, MEV protection). Deployment requires only a standard smart contract with timestamp-based expiry.

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:

- **Cryptographic honeypots:** Hiding embedded secrets that trigger rewards
- **MEV-resistant execution:** Preventing sandwich attacks on trade logic
- **Sealed-bid auctions:** Hiding bid evaluation until auction completion
- **Private liquidation thresholds:** Obfuscating conditions that trigger liquidation
- **Dark pool matching:** Order matching resistant to reverse-engineering

These applications share a common property: *eventually-expiring secrets.* Once an auction settles, a honeypot is sprung, or an order is executed, learning the underlying secret no longer confers advantage. This stands in contrast to long-lived secrets such as FHE decryption keys or static signing keys, where compromise at any future time is catastrophic. TLO is explicitly designed for the former category.

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 [8, 7, 4], but with overhead exceeding $10^6\times$—far too expensive for on-chain deployment.

1.1 Our Approach: Two-Layer Obfuscation

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

Two-Layer Security Model:

1. **Layer 1 (Topology):** Structural mixing defeats structural and statistical attacks through wire selection patterns (non-pow2 distances, uniform wire usage, irregular layers). *Security: heuristic, empirically validated.*
2. **Layer 2 (LWE):** On-chain inner product computation defeats semantic attacks (RainbowTable). Control functions hidden via LWE ciphertexts. *Security: computational, based on* LWE *hardness (see Table 7.2 for actual estimates).*

Key mechanism: Wrong-key decryption yields garbage—attackers can simulate offline but learn nothing without the correct secret.

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 LWE ciphertexts 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 LWE

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 on-chain contract reinforces this by exposing only a 1-bit oracle (true/false), limiting information leakage per query.

1.3 Contributions

1. **TLO Framework:** A two-layer obfuscation framework combining topology-based structural mixing with on-chain LWE inner products, achieving practical gas costs (∼2.58M for n=64) with 6/6 attack resistance in our 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 on-chain 1-bit oracle limits information leakage per query.

4. **On-Chain LWE:** Control functions hidden via LWE ciphertexts with full inner product computation on-chain. No hints or shortcuts—true LWE hardness.

5. **Post-Quantum Resistance:** LWE-based security with no cryptographic assumptions in the topology layer. See Table 7.2 for security estimates by dimension.

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

1.4 Scope and Limitations

What TLO provides:

- Computational security based on LWE hardness (see Table 7.2 for estimates: ∼108-bit classical for n=64 with uniform secrets) plus empirical heuristics (topology layer)
- Attack resistance (6/6 in our evaluation matrix—not a universal security claim)
- Practical gas costs (∼2.58M for n=64, 8.6% of block limit)
- Post-quantum resistance (relative to LWE parameters)
- 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 TLO 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 6-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 solving 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 [5] established iO from multilinear maps. Recent constructions achieve iO from LWE [8, 7, 4]. Diamond iO [12] 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 [13] and Goyal, Koppula, and Waters [6] achieve VBB security for compute-and-compare programs (predicates of the form $P(x) = [f(x) = s]$) under LWE. Our application of C&C to control function hiding is inspired by this work.

Local Mixing. Canetti et al. [3] 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., Tornado Cash for private transfers) or Trusted Execution Environments (e.g., Secret Network). TLO provides a new point in the design space: on-chain obfuscation without trusted hardware or external proving infrastructure.

2 Application Scope

2.1 Valid Applications: Eventually-Expiring Secrets

TLO is intended for applications where the embedded secret or predicate has a natural *expiration time*: once a certain event occurs, learning the secret is harmless. Examples include:

- **Cryptographic honeypots:** The reward condition is burned once triggered. After claiming, the secret has no value.
- **Sealed-bid auctions:** Bids and evaluation logic are revealed at settlement. Post-settlement analysis is harmless.
- **Lotteries and prediction markets:** Outcomes are revealed after close. Knowing the winning condition afterward confers no advantage.
- **MEV protection:** Relevant order flow is short-lived. By the time an attacker analyzes the circuit, the trade has executed.
- **Dark pool matching:** Trade conditions expire quickly. Historical analysis doesn't affect future trades.

2.2 Invalid Applications: Long-Term Keys

TLO is *not* designed to protect secrets that must remain confidential indefinitely:

- Long-term FHE decryption keys or signing keys
- Static master keys or decryption policies shared across many contracts
- Liquidation thresholds or risk parameters that remain valid indefinitely

For such long-lived secrets, an adversary can analyze the obfuscated circuit for arbitrarily long time. While C&C/LWE provides computational security, we do not claim that this is sufficient for secrets requiring decades of protection. Standard key management (HSMs, threshold schemes) or full iO should be used instead.

3 Preliminaries

3.1 Learning With Errors

Definition 3.1 (LWE [11])**.** For dimension n, modulus q, and error distribution χ, the $\mathsf{LWE}_{n,q,\chi}$ 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 [11, 10])**.** *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 [9]. Note that our parameters (n=64, q=65521) are *much smaller* than NIST-standardized profiles (which use $n \geq 512$). See Table 7.2 for security estimates based on the LWE Estimator [1].

3.2 Compute-and-Compare Obfuscation

Definition 3.3 (C&C [6, 13])**.** 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 [13])**.** *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: $\mathit{state}'[a] = \mathit{state}[a] \oplus c_f(\mathit{state}[c_1], \mathit{state}[c_2])$

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 TLO Framework

4.1 System Architecture

TLO consists of two 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 inner product computation hides control functions, defeating semantic attacks by preventing subcircuit evaluation.

Deployment: A single smart contract with:

- Circuit data (gates encoded as bytes)
- 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.

- **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 LWE (see Table 7.2 for security estimates).
- **Expiry:** After the timestamp, the secret may no longer be valuable (application-specific).

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
4. Deploy contract with circuit data (SSTORE2), expected output hash, and expiry timestamp

Solving (Bob):

1. Compute $H = \mathsf{hash}(\mathsf{sender} \| \mathsf{guess})$
2. Call `commit(H)`
3. Wait $k \geq 1$ blocks
4. Call `reveal(guess)`
5. If $C(\mathsf{guess}) = 1$, receive reward

Attack (Charlie):

1. Analyze circuit structure (topology layer blocks structural attacks)
2. Attempt subcircuit evaluation (LWE layer blocks—CFs hidden by LWE)
3. Resort to brute force (2^{256} for 256-bit secrets—infeasible)

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 defeats structural and statistical attacks:

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 defeats structural and statistical attacks.

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 defeat known attack heuristics.

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. LWE cryptography provides computational hardness.

On-Chain Inner Product: Each gate's control function is encoded as an LWE ciphertext (a, b) where $b = \langle a, s \rangle + e + \mathsf{bit} \cdot q/2$. 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), TLO stores full (a, b) ciphertexts. Recovering CFs requires solving LWE.

Proposition 6.1 (LWE Hides Subcircuit Semantics)**.** *Under* LWE *hardness, 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 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.

Under LWE hardness, recovering the correct CF bits from (a, b) without knowing s_{enc} is computationally infeasible.

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

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, 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: $n \in \{16, 32, 64\}$ (see Table 7.2 for security estimates)
- Error: Gaussian with $\sigma = 1024$ (safe since $\sigma \ll q/4 = 16380$)

Storage per gate: $n \cdot 2 + 2$ bytes per ciphertext, 4 ciphertexts per gate, plus 3 bytes for pins.

- n=16: 139 bytes/gate, 89 KB total (640 gates)
- n=32: 267 bytes/gate, 171 KB total
- n=64: 523 bytes/gate, 327 KB total

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 Two-Layer Security Model

TLO provides security through complementary layers with different security bases:

1. **Topology layer (heuristic):** Defeats structural and statistical attacks through wire selection patterns that break attack heuristics. Security is *empirical*—validated against our attack suite but not formally proven.
2. LWE **layer (computational):** Defeats semantic attacks by hiding control functions via LWE ciphertexts with on-chain inner products. Security is *computational*—based on LWE hardness (see Table 7.2).
3. **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. A key property of TLO 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 standard LWE assumptions. This arises naturally from expanding a cryptographic hash output to n elements modulo q. The uniform-secret variant is *harder* to attack than small-secret LWE: primal (uSVP) attacks rely on finding short vectors (s, e) in a lattice, but when s has norm $\sim\sqrt{n} \cdot q/2$ rather than $\sim\sqrt{n}$, the attack fails. We validated this empirically: BKZ-50 reduction on n=16 instances failed to recover the secret after 200+ basis updates.

Table 7.2 presents security estimates for uniform-secret LWE based on dual-attack analysis (which remains effective regardless of secret distribution). These estimates use BKZ lattice reduction with the "Core-SVP" cost model.

Dimension n	**Classical**	**Quantum**	**Gas**	**Notes**
n=16	∼22-bit	∼20-bit	744K	Toy parameters
n=32	∼51-bit	∼46-bit	1.27M	Low security
n=64	∼108-bit	∼98-bit	2.58M	Short-lived secrets
n=96	∼178-bit	∼162-bit	∼4M	Medium-lived secrets
n=128	∼203-bit	∼185-bit	∼5M	Long-lived secrets
n=256	∼132-bit	∼120-bit	∼10M	NIST-level security

Table 2: LWE security estimates for uniform-secret variant (q=65521, σ=1024). Validated with lattice-estimator. Uniform secrets resist primal attacks; estimates based on dual attack. Gas costs are for 64-wire/640-gate circuits.

Important: Our default parameters (n=64) provide ∼108-bit classical security. This is suitable for *eventually-expiring secrets* with lifetimes of hours to days (e.g., MEV windows, auction rounds). For longer-lived secrets, use $n \geq 128$ (∼203-bit) or n=256 for NIST-level security. We validated these estimates via direct lattice attacks using fpylll/BKZ; see §9.1.

7.3 Security Definitions

Definition 7.1 (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 7.2 (LWE Control Function Security)**.** Control function encoding is secure if no PPT adversary can recover CF bits from LWE ciphertexts with non-negligible advantage under LWE hardness.

7.4 Assumptions

Assumption 7.3 (LWE Hardness)**.** *The Learning With Errors problem is computationally hard for the chosen parameters.*

Assumption 7.4 (Topology Empirical Security)**.** *The topology layer's wire selection algorithm defeats the structural and statistical attacks in our evaluation matrix. This is validated empirically, not proven.*

7.5 Main Theorem

Theorem 7.5 (TLO Attack Resistance). *Under Assumption 7.3 (*LWE *hardness), Assumption 7.4 (topology empirical security), and the wrong-key-gives-garbage property (1-bit on-chain oracle with no subcircuit access), TLO achieves extraction resistance against our 6-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, breaking these patterns. *Empirically validated* (Assumption 7.4).

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. *Empirically validated* (Assumption 7.4).

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 7.3).

Therefore, all 6 attack classes in our matrix are defeated under the stated assumptions. *This is not a claim of universal security.* □

7.6 Post-Quantum Security

Proposition 7.6 (TLO Post-Quantum Security). *TLO is post-quantum secure assuming* LWE *is quantum-resistant.*

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

The lattice layer uses LWE, which is believed quantum-resistant and forms the basis of NIST post-quantum standards [9].

Therefore, TLO inherits the quantum resistance of LWE. □

7.7 Oracle Model Considerations

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

Observation 7.7. *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 7.8 (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).

8 Gas Costs and Deployment

8.1 Gas Estimates

Storage: Circuit data is stored via SSTORE2 (code-as-storage):

LWE **Dimension**	**Security (Classical)**	**Gas**	**Block %**
n=16	∼22-bit	744K	2.5%
n=32	∼22-bit	1.27M	4.2%
n=64	∼108-**bit**	**2.58M**	**8.6%**

Table 3: TLO gas costs by LWE dimension (64-wire/640-gate circuits, Tenderly-confirmed). Security estimates for uniform-secret LWE (dual attack).

- n=16: 89 KB
- n=32: 171 KB
- n=64: 335 KB

For the paper, we use n=64 providing ∼108-bit classical security—suitable for eventually-expiring secrets (hours to days).

9 Attack Evaluation

We empirically validated TLO 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
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 4: 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 TLO 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.

9.1 Lattice Attack Validation

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

Experimental setup:

- Generated TLO 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=64:** Attack running for 9+ hours on dedicated server (still in progress at time of writing). Extended runtime validates that $\sim$108-bit security is realistic.

Key finding: The uniform-secret variant of LWE used in TLO 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/tlo` (scripts/lattice_attack/).

10 Implementation

10.1 Smart Contract Architecture

TLO requires no external infrastructure—just a standard smart contract:

Listing 1: TLOHoneypot Contract (simplified)

```
contract TLOHoneypot {
    bytes public circuitData;
    uint256 public secretExpiry;
    bytes32 public expectedOutputHash;

    mapping(address => bytes32) public commits;

    function check(bytes32 input) external view returns (bool) {
        require(block.timestamp < secretExpiry, "Expired");
        return evaluate(input) == expectedOutputHash;
    }

    function commit(bytes32 h) external {
        commits[msg.sender] = h;
    }

    function reveal(bytes32 secret) external {
        require(keccak256(abi.encode(secret, msg.sender))
                == commits[msg.sender], "Bad commit");
        require(evaluate(secret) == expectedOutputHash, "Wrong");
        // Transfer reward
    }
}
```

10.2 Gas Costs

Operation	**Gas (n=64)**	**% of 30M limit**
`check()`	2.58M	8.6%
`commit()`	47K	0.16%
`reveal()`	2.61M	8.7%

Table 5: Gas costs on 64-wire/640-gate circuits with n=64 LWE dimension.

10.3 Commit-Reveal Protocol

To prevent front-running:

1. **Commit:** User submits $H = \mathsf{hash}(\mathsf{sender}\|\mathsf{secret})$
2. **Wait:** $k \geq 1$ blocks
3. **Reveal:** User submits secret; contract verifies H matches

Sender binding prevents reward theft: an attacker copying the reveal transaction computes $\mathsf{hash}(\mathsf{attacker}\|\mathsf{secret}) \neq H$.

11 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 TLO's 2.58M gas for ∼26-bit LWE security?

11.1 When Hash-Compare Suffices

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

- Lower gas (∼50K vs 2.58M)
- Higher security (256-bit vs 26-bit)
- 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.

11.2 When TLO Adds Value

TLO 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, TLO 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—TLO’s wrong-key garbage property ensures all wrong inputs look equally random.
4. **Composable obfuscation:** TLO circuits can be composed. Multiple predicates can share structure without revealing which checks are related.

11.3 Multi-Bit Output: The Key Distinction

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

Approach	**Output**	**What’s Hidden**
Hash-compare	1 bit (true/false)	Secret value only
TLO 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.

TLO 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
}

// TLO: extracts hidden parameter on match
function extractThreshold(bytes32 x) returns (int256) {
    uint256 output = evaluateTLO(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.

11.4 Point Function with Hidden Payload

Both hash-compare and TLO 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")
- **TLO:** Point function with N-bit output ("here's the hidden result")

For applications where the *payload* matters (not just the confirmation), TLO provides value that hash-compare cannot. The 57× gas premium buys multi-bit hidden computation, not stronger unlocking security.

11.5 Honest Assessment

For most honeypot use cases ("find the 256-bit secret"), hash-compare is the right choice. TLO'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 TLO is always better than hash-compare. The choice depends on the application's threat model and whether multi-bit hidden output is required.

12 Limitations and Open Problems

12.1 Theoretical Gaps

1. **Topology security is empirical:** We validate against our 6-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. TLO 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.

12.2 Practical Concerns

1. **Gas costs:** TLO with n=64 LWE requires ~2.58M gas (8.6% of block limit). This may be prohibitive for some applications. Lower security configurations (n=16, 744K gas) are available for cost-sensitive use cases.
2. **Storage:** Circuit data requires 335 KB (SSTORE2) for n=64. Deployment costs are dominated by SSTORE2 writes.
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.

12.3 What TLO 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 [2].

4. **Universal Security:** We resist our 6-class taxonomy, not all possible attacks.

13 Conclusion

TLO provides practical circuit obfuscation for smart contracts through two-layer defense:

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

- LWE **layer (computational):** On-chain inner product computation hides control functions, defeating semantic attacks. ∼108-bit classical security with n=64 using uniform secrets (see Table 7.2).

TLO achieves 6/6 resistance against our attack evaluation matrix at ∼2.58M gas (n=64, 8.6% of block limit). Security is based on LWE hardness combined with empirical topology properties. The system targets *eventually-expiring secrets* where moderate security suffices for the secret's lifetime. Requires only a standard smart contract with timestamp expiry.

Key contributions:

- On-chain LWE inner products for true control function hiding

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

- Practical gas costs within EVM block limits

- Honest comparison with hash-compare baseline (Section 11)

Limitations: Topology security is heuristic, not proven. Gas costs may be prohibitive for some applications.

Future Work:

- Formal analysis of topology layer security

- Multi-bit output extensions

- Further gas optimizations

References

[1] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. Journal of Mathematical Cryptology, 2015. LWE Estimator: `https://bitbucket.org/malb/lwe-estimator`.

[2] B. Barak, O. Goldreich, R. Impagliazzo, S. Rudich, A. Sahai, S. Vadhan, and K. Yang. On the (im)possibility of obfuscating programs. *Journal of the ACM*, 59(2), 2012.

[3] R. Canetti, C. Chamon, E. R. Mucciolo, and A. E. Ruckenstein. Towards general-purpose program obfuscation via local mixing. Cryptology ePrint Archive, Report 2024/006, 2024.

[4] M. Cini, R. W. F. Lai, and G. Woo. NTRU + equivocal LWE iO. In *CRYPTO*, 2025.

[5] S. Garg, C. Gentry, S. Halevi, M. Raykova, A. Sahai, and B. Waters. Candidate indistinguishability obfuscation from multilinear maps. In *FOCS*, 2013.

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

[7] R. Goyal and Y. Polychroniadou. Indistinguishability obfuscation from well-founded assumptions. In *STOC*, 2021.

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

[9] National Institute of Standards and Technology. Module-lattice-based key-encapsulation mechanism standard (ml-kem). `https://doi.org/10.6028/NIST.FIPS.203`, 2024. Standardizes ML-KEM, based on CRYSTALS-Kyber.

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

[11] O. Regev. On lattices, learning with errors, random linear codes, and cryptography. In *STOC*, 2005.

[12] S. Suegami, E. Bottazzi, and G. Park. Diamond iO: A straightforward construction of indistinguishability obfuscation from lattices. Cryptology ePrint Archive, Report 2025/236, 2025. Privacy & Scaling Explorations, Ethereum Foundation.

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

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, $m = 640$ gates) using our implemented attack suite:

Attack	**TLO** (n=64)
Compression	0%
PatternMatch	0%
Structural	0%
Statistical	0%
DiagonalCorrelation	0%
RainbowTable	0%

Table 6: 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.