TLOS: Topology-Lattice Obfuscation for Smart Contracts

Anonymous Submission

Abstract

How can we protect low-entropy secrets—like human-memorable passwords or MEV bot triggers—on a fully transparent blockchain? We propose TLOS, a lightweight obfuscation framework that secures sensitive predicates against GPU brute-force attacks. By layering standard LWE cryptography with a "planted puzzle" proof-of-work, TLOS forces attackers to spend significant computation per guess (2^{76} search space), while legitimate verification remains cheap (1.8M–6M gas for n=384, depending on circuit size). This approach bridges the gap between theoretical obfuscation and practical deployment, enabling new classes of privacy-preserving applications on Ethereum Mainnet.

1 Introduction

Smart contracts are fully transparent. This conflicts with applications requiring hidden logic: honeypots, MEV-resistant execution, sealed-bid auctions, and private thresholds.

Indistinguishability obfuscation (iO) from lattices [1] achieves strong security but with impractical overhead. We take a different approach: *practical obfuscation* that resists known attack classes through four complementary layers:

Four-Layer Defense-in-Depth:

1. **Topology (stops the reverse engineer):** Structural mixing raises the bar against pattern-matching. *Security: heuristic (hard security from Layers 2 & 4).*
2. **LWE (stops the cryptanalyst):** On-chain inner products hide control functions. *Security:* $\sim 2^{112}$ *PQ for* n=384.
3. **Wire Binding (stops the hacker):** Full-rank linear hash prevents mix-and-match. *Security: algebraic binding.*
4. **Puzzle (stops the GPU miner):** Forces $3^{48} \approx 2^{76}$ brute-force work. *Security: computational.*

1.1 Contributions

1. **TLOS Framework:** Four-layer obfuscation for EVM using lattice assumptions (1.8M–6M gas with n=384).
2. **Full-Rank Wire Binding:** Inter-gate wire binding via on-chain 64×64 matrix-vector products. Inspired by [1] but simplified: a public bijective map providing algebraic binding, not cryptographic hiding.
3. **Per-Batch Updates:** Gas-efficient binding (128-gate batches, 5 updates for 640 gates).
4. **Layer 4 Puzzle:** Planted LWE puzzle (n=48, m=72, q=2039) forces 2^{76} minimum search space at 1.26M gas overhead.
5. **PRG Optimization:** Derive 16 matrix coefficients per keccak call (320 calls vs 4096).
6. **Use Case Analysis:** When circuit obfuscation provides advantage over simple hash commitments.

2 Advantage Over Hash Commitments

A natural question: why not simply use `keccak256(secret)`? For random 256-bit secrets, keccak is indeed simpler. However, TLOS excels for *low-entropy secrets* and *multi-bit payloads*.

Secret Type	Keccak	TLOS
Random 256-bit	2^{256} hashes	$\min(2^{256}, 2^{112})^{\dagger}$
Human password (10^6)	<1 ms	∼10 sec
Range 0–100K	<0.1 ms	∼1 sec
4-word phrase (2048^4)	∼18 sec	∼5 months

Table 1: Offline brute-force cost (GPU). †For high-entropy secrets, TLOS security is bounded by LWE hardness ($\sim 2^{112}$). Assumptions: keccak at 10^9 hashes/s (GPU), TLOS at 10^5 guesses/s (GPU offline simulation).

2.1 Deployment Constraints

TLOS is designed for environments where:

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

- **No zk-SNARKs:** Proof systems unavailable or too complex
- **No private L2:** Must deploy on public L1
- **Gas is cheap:** At 0.03 gwei, 2M gas (128 gates) $\approx$ \$0.19

Under these constraints, TLOS provides a practical way to make low-entropy secret verification expensive on EVM—no memory-hard KDF (Argon2/scrypt) exists as an EVM precompile.

2.2 Why Not Keccak?

Low-entropy secrets. Keccak can be brute-forced at $\sim$1ns/guess (10^9/s on GPU). TLOS transforms each guess into an expensive LWE inner-product computation ($\sim 10\mu$s/guess, or 10^5/s on GPU), a $10^4\times$ slowdown.

Multi-bit payloads. With keccak, a payload requires separate ciphertext $\mathsf{Enc}_k(\mathsf{payload})$ alongside $\mathsf{keccak256}(k)$. Attackers who crack k decrypt offline. In TLOS, payload is *fused* into the circuit.

Hidden structure. Keccak reveals what you're checking ("it's a hash comparison"). TLOS hides the check structure—attackers don't know if it's one condition or ten.

2.3 Practical Use Cases

DeFi:

- *Hidden liquidation thresholds:* MEV bots can't front-run if they don't know your trigger price
- *Private stop-loss/take-profit:* Strategy parameters hidden inside circuit
- *Dark limit orders:* Price bands obfuscated from sandwich attacks

Wallet Recovery & Inheritance:

- *Human-memorable recovery codes:* 6-word phrase protects wallet; keccak would be instant to crack, TLOS takes hours
- *On-chain wills:* Family passphrase unlocks inheritance; structure (who gets what) is hidden
- *Dead man's switch:* Liveness check + secret claim code, both obfuscated

Gaming & NFTs:

- *On-chain puzzles:* Low-entropy answers protected from GPU brute-force
- *Sealed-bid auctions:* Bid values from small domain can't be dictionary-attacked
- *Mystery box traits:* Generation logic hidden; can't predict rare mints

Identity & Access Control:

- *ENS recovery:* Human-memorable codes protect high-value names
- *One-time delegation codes:* 6-digit codes; TLOS makes 10^6 guesses take hours
- *Multi-code access:* OR of N event codes with hidden structure

When to use keccak: Random 256-bit secrets with no payload and no structure to hide.

3 Construction

3.1 Full-Rank Wire Binding

We bind wire values across gates using a *full-rank* linear hash: $H(x) = Ax \mod q$ where $A \in \mathbb{Z}_q^{64\times 64}$ is derived from `circuitSeed` and gate index. This is inspired by [1] but is *not* subspace-evasive in the formal sense—it is simply a public bijective linear map providing algebraic binding.

Why 64×64? A full-rank 64×64 matrix over $\mathbb{Z}_q$ has trivial kernel (only the zero vector), so H is a bijection on $\mathbb{Z}_q^{64}$: every accumulator state has a unique preimage. The matrix is derived via PRG expansion from `circuitSeed`; for random A over $\mathbb{Z}_q$, full-rank probability is $> 1-10^{-19}$.

Threat model (mix-and-match): Without wire binding, an adversary could evaluate different branches of the circuit and splice together wire values from distinct execution paths to synthesize a "fake" trace. The full-rank hash forces all wires in a batch to be combined into a 1024-bit accumulator; any deviation changes the accumulator in a way that cannot be repaired without recomputing a full, consistent trace.

Note: This is a public random linear map over $\mathbb{Z}_q$ (no noise, no secret). Security is *integrity/binding*, not cryptographic collision resistance: once an accumulator value is fixed, there is (for full-rank A) a single consistent wire vector that can produce it.

```
// Optimized: 16 coefficients per keccak (320 calls vs 4096)
function _wireBindingHash(uint256 input, uint256 gateIdx)
    internal view returns (uint256[4] memory output) {
  assembly {
    for { let row := 0 } lt(row, 64) { row := add(row, 1) } {
      let rowSeed := keccak256(seed, gateIdx, row) // 1 per row
      let sum := 0
      for { let blockIdx := 0 } lt(col, nWires) { ... } {
        let blockDigest := keccak256(rowSeed, blockIdx) // 16
             coefs
        // Extract 16 u16 coefficients from blockDigest
        for { let k := 0 } lt(k, 16) { k := add(k, 1) } {
          let aij := mod(and(shr(mul(k,16), blockDigest), 0
               xFFFF), q)
          if bitVal { sum := add(sum, aij) }
        }
      }
      // Pack into output
    }
  }
}
```

3.2 Per-Batch Updates

For gas efficiency, the consistency hash is updated every 128 gates:

$$\mathsf{hashAcc}_{k+1} = H(\mathsf{hashAcc}_k \oplus \mathsf{wires}, \mathsf{batchEnd}_k)$$

This provides 5 hash updates for 640 gates. Combined with the PRG optimization (16 coefficients per keccak instead of 1), total wire binding overhead is reduced by $\sim$25$\times$ compared to naive implementation.

3.3 Planted LWE Puzzle (Layer 4)

Layers 1–3 protect against structural, semantic, and mix-and-match attacks, but cannot prevent GPU brute-force over low-entropy inputs. If the secret has only 2^{30} entropy, an attacker evaluating at 10^6 guesses/sec cracks it in $\sim$17 minutes.

Layer 4 forces a minimum search space by requiring solvers to find a ternary secret $s \in \{-1, 0, 1\}^{48}$ satisfying:

$$\|As - b\|^2 < \tau$$

where (A, b) are deterministically derived from the input x. The puzzle uses a *planted* construction: for each x, we derive a planted secret s^* and error e, then set $b = As^* + e \mod q$. The solver must recover s^* (or an equivalent short solution).

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

Security: GPU brute-force at 436M guesses/sec (GH200) requires $\sim$5.7 million years for exhaustive search. Even 10,000 GPUs ($\sim$570 years) cannot crack in practical time.

Integration: The puzzle solution hash $H(s)$ is combined with the input to derive the LWE secret, providing double binding: attackers must solve the puzzle AND have the correct input.

3.4 LWE Parameterization

Our control-function hiding uses standard LWE with Gaussian noise:

Parameter	Value
Dimension n	384
Modulus q	65521 (largest 16-bit prime)
Sample count m	2560 (4 ciphertexts $\times$ 640 gates)
Secret distribution	Uniform over $\mathbb{Z}_q^n$ (Keccak-derived)
Public vector a	Seed-derived over $\mathbb{Z}_q^n$ (11 bytes/gate)
Error distribution	Gaussian with σ=8

Encoding: Encryption uses standard LWE: $b = \langle a, s\rangle + e + \mu \cdot (q/2) \mod q$, where $\mu \in \{0, 1\}$ encodes the control-function bit and $e \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise. Decryption uses threshold $q/4$: if $(b - \langle a, s\rangle) \mod q$ falls in $(q/4, 3q/4)$, output 1; else 0.

Correctness: With σ=8 and threshold at $q/4 \approx 16380$, decryption error probability is negligible ($< 2^{-40}$). The Gaussian noise provides semantic security while maintaining practical correctness.

Security estimate: Using the lattice estimator [4] for LWE(n=384, q=65521, σ=8, m=2560), we obtain $\sim 2^{112}$ post-quantum security. This follows standard LWE hardness assumptions with reductions to GapSVP/SIVP.

LWE Hardness: Our construction uses standard LWE: $b_i = \langle a_i, s\rangle + e_i + \mu_i \cdot (q/2) \mod q$ where $e_i \sim \mathcal{N}(0, \sigma^2)$ with σ=8. The $\sim 2^{112}$ post-quantum estimate is based on the lattice estimator [4] using the dual attack with BKZ/sieving cost model. Worst-case to average-case reductions [2] justify LWE as a hardness assumption, but our concrete security figure is based on best-known algorithms, not the reductions themselves.

Validation: We implemented 9 attack classes (`scripts/tlos_attack.py`) to verify correctness and confirm no obvious breaks exist. These exploratory tests are *not* a substitute for the estimator-based security analysis; they only confirm the implementation is sound.

4 Security

Layer	Basis	Security
CF hiding	LWE (n=384, σ=8)	$\sim 2^{112}$
Wire binding	64×64 linear	Algebraic
Topology	Heuristic	Empirical
Puzzle	Ternary LWE (n=48)	2^{76} min

Table 2: Security profile. LWE with Gaussian noise (σ=8).

We estimate $\sim 2^{112}$ post-quantum cost against the underlying LWE instances using the lattice estimator for LWE(n=384, q=65521, σ=8, m=2560). This follows standard LWE hardness with reductions to GapSVP/SIVP (see §3.4). This security level is suitable for:

- Medium-term secrets (weeks–months)
- Production applications with moderate value
- Real-world deployment scenarios

For even higher security margins, $n \geq 512$ could be considered with modest gas increase.

4.1 Combined Security Model

Security relies on *both* layers working together:

- **Topology** hides μ (control function bits) from structural analysis
- **LWE** hides s given encrypted μ values
- **Wire Binding** ensures execution trace consistency (integrity, not hiding)
- **Puzzle** forces minimum work per guess (2^{76})

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

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

where 2^h is brute-force over inputs, 2^{112} is LWE security, and 2^{76} is the puzzle's per-guess cost floor. Layers 1 and 3 raise the bar against structural/mix-and-match *shortcuts* but do not multiply the security level.

The $\sim 2^{112}$ estimate assumes μ cannot be predicted from circuit structure. If an attacker predicts μ for n=384 gates via structural analysis, they can recover s via Gaussian elimination in $O(n^3)$.

Why structural attacks fail: TLOS circuits compute point functions ($C(x)$=1 iff x=secret). There are no adders, comparators, or recognizable gadgets—only reversible permutations with one distinguished output.

Input-dependent key: The decryption key $s=H(\mathsf{input})$ is derived at evaluation time. Tracing with wrong input x' yields random diff values, revealing nothing about μ. *Traces are visible but useless without the correct input.*

4.2 What Wire Binding Provides

- **Mix-and-match detection:** Any attempt to evaluate gates with inconsistent inputs will, except with negligible probability, yield a final accumulator that does not match the expected value.
- **Execution trace binding:** Full evaluation history is committed.
- **Lattice-style construction:** Uses the same linear algebra as the LWE layer for design uniformity; its security role is *integrity/binding*: for a fixed full-rank A, each accumulator state has a unique preimage in $\mathbb{Z}_q^{64}$ under H.

4.3 Limitations

TLOS does *not* provide:

- Semantic security against black-box evaluation (not targeted; we resist specific attack classes only)
- Full iO (indistinguishability obfuscation) (not targeted)

5 Implementation

Contracts: `TLOSLWE.sol` (~350 lines Solidity)

Rust generator: Produces circuit data matching Solidity evaluation exactly. Storage is 11 bytes/gate via seed-derived a vectors (vs. explicit storage).

Gas costs (measured, n=384, LWE + wire binding only):

- **64 gates**: ~1.8M gas (3% of block)
- **128 gates**: ~2M gas (3% of block)
- **256 gates**: ~3M gas (5% of block)

Full 4-layer (add puzzle overhead): +1.26M gas for Layer 4 puzzle verification. Example: 128 gates full = ~3.3M gas (5.5% of block).

Optimizations applied:

- PRG optimization: 16 coefficients per keccak (320 calls vs 4096 per update)
- Single `mod` at end of inner product (vs per-term)
- Batch size 128 (5 binding updates for 640 gates)
- n=384 LWE dimension for $\sim 2^{112}$ PQ security

6 Topology Mixing Experiments

We empirically evaluated topology mixing against simple structural attacks on 100 random 64-wire/640-gate circuits.

Attacker model: Adversary sees the full obfuscated circuit graph (wire indices, gate structure) but not the LWE-encrypted control functions. Goal: identify semantic structure (e.g., comparators, adders) or reduce circuit complexity.

Attacks tested:

- **Degree analysis:** Identify high-fanout "hub" wires that might carry semantic signals.
- **Pattern matching:** Search for known subcircuit patterns (XOR chains, comparator structures).
- **Wire distance heuristic:** Exploit power-of-2 distances between control wires.

Results:

Attack	Baseline	With Mixing
Hub wire identification	78%	12%
Pattern match (XOR chain)	91%	8%
Power-of-2 distance exploit	100%	0%

Topology mixing uses non-power-of-2 wire distances and uniform wire usage, which eliminates the power-of-2 distance exploit entirely and significantly reduces hub/pattern detection rates. These results are empirical; we do not claim formal security.

6.1 Layer Ablation

We evaluated attack resistance with different layer combinations:

Configuration	Struct.	Semantic	Mix-Match	Brute-Force
Baseline (no obfuscation)	0/3	0/1	0/1	0/1
Topology only	3/3	0/1	0/1	0/1
Topology + LWE	3/3	1/1	0/1	0/1
Topology + LWE + Binding	3/3	1/1	1/1	0/1
Full 4-layer (+ Puzzle)	3/3	1/1	1/1	1/1

Key findings: (1) Topology alone defeats structural attacks but not semantic (RainbowTable) attacks. (2) LWE is necessary to hide control functions from semantic analysis. (3) Wire binding is necessary to prevent mix-and-match—without it, an attacker can evaluate gates with inconsistent wire values. (4) The planted LWE puzzle forces minimum 2^{76} brute-force work regardless of input entropy, protecting low-entropy secrets from GPU attacks.

7 Conclusion

TLOS is a practical circuit obfuscation framework for EVM using lattice-based assumptions, achieving $\sim 2^{112}$ post-quantum security at 1.8M–6M gas with n=384 and full-rank wire binding. The four-layer design (topology + LWE control function hiding + full-rank wire binding + planted LWE puzzle) provides resistance against structural, statistical, semantic, mix-and-match, and brute-force attacks.

Key insight: TLOS provides significant advantage over simple hash commitments for *low-entropy secrets* by making each guess computationally expensive. Its value is in making low-entropy guesses expensive under EVM constraints and enabling richer obfuscated logic.

Future work includes even higher security parameters ($n \geq 1024$), succinct verification via zkSNARK/STARK, and formal security analysis of batch wire binding.

References

[1] Ma, Dai, Shi. "Indistinguishability Obfuscation from Lattices in the Quantum Random Oracle Model." ePrint 2025/307.

[2] Regev. "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography." STOC 2005.

[3] Goyal, Koppula, Waters. "Lockable Obfuscation." FOCS 2017.

[4] Albrecht, Player, Scott. "On the Concrete Hardness of Learning with Errors." J. Math. Cryptol. 2015.