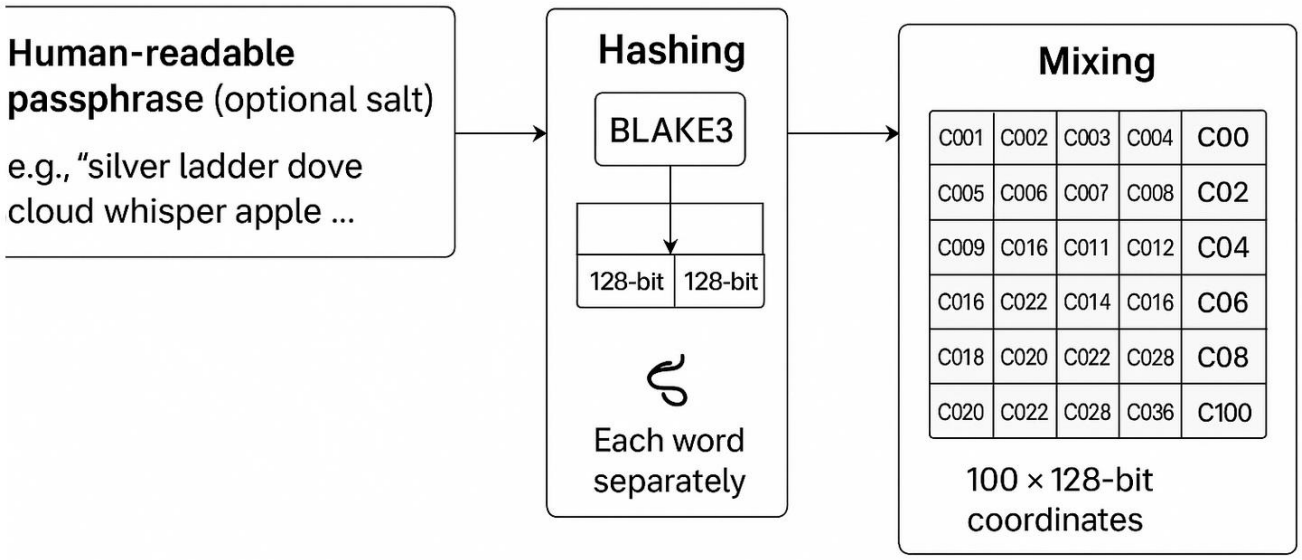
Quantum-Resistant Coordinate Derivation from Natural Language Passphrases with Hash-Based Post-Quantum Signatures

Authors: @Cryptis – Cryptix-Network.org
Date: 2025-10-13
Submission to: Institute of Commercial Cryptography Standards (ICCS), Next-generation Commercial Cryptographic Algorithms Program
Licence: MIT License

Abstract



We propose a deterministic, quantum-resistant method for deriving high-entropy numerical coordinates from natural language passphrases. The scheme outputs 100 × 128-bit values (12,800 bits total) via per-word cryptographic hashing combined with a multi-round nonlinear mixing function. The resulting coordinates serve as a versatile seed for post-quantum hash-based signature schemes, offering an asymmetric, hash-only approach robust against both classical and quantum adversaries. This method achieves practical security without memory-hard computations or algebraic hardness assumptions, making it suitable for constrained environments and post-quantum applications.

---

1. Introduction

Current cryptographic systems face increasing threats from quantum computing. Shor's algorithm compromises traditional public-key systems such as RSA and ECC, while Grover's algorithm provides a quadratic speedup for unstructured search attacks. To address these

threats, we present a minimalist, hash-based framework to deterministically derive cryptographic coordinates from human-readable passphrases. These coordinates can be used for secure seed generation, post-quantum key derivation, and as inputs for hash-based signature schemes such as SPHINCS+ or XMSS.

---

2. Methodology

2.1 Overview

- **Input:** A natural language passphrase, optionally with a salt.
- **Output:** 100 deterministic, high-entropy coordinates (128-bit each, total 12,800 bits).
- **Approach:**
    1. Hash each word separately using a modern cryptographic hash function (e.g., BLAKE3).
    2. Split each 256-bit digest into two 128-bit values.
    3. Apply a custom nonlinear mixing function over multiple rounds to generate several coordinates per word.
    4. Distribute the coordinates evenly across all words to achieve a total of 100 outputs.

This design ensures high diffusion, nonlinearity, and irreversibility, producing an astronomically large output space.

2.2 Nonlinear Mixing Function

The mixing function applies:

- Bit rotations
- Multiplicative scramblings
- XORs
- Additive round constants

This combination guarantees resistance to preimage attacks and uniform distribution of outputs.

2.3 Hash Functions

While the demonstration uses **BLAKE3**, the scheme is hash-agnostic and compatible with any modern cryptographic hash function.

---

3. Quantum Security Considerations

Threat	Impact on Scheme	Mitigation
Shor's Algorithm	Efficiently factors integers and computes discrete logarithms	Not applicable: scheme relies on hash-based and nonlinear bitwise operations

Threat	Impact on Scheme	Mitigation
Grover's Algorithm	Quadratic speedup for brute-force search	Security reduced from 2^12800 to ~2^6400; still astronomically large

The extreme output width ensures practical security even in the presence of large-scale quantum computers.

---

4. Salt Usage and Security

- Optional salt per word increases randomness and prevents rainbow table attacks.
- Not strictly required due to vast output space, but recommended in multi-system or multi-user environments.

---

5. Comparison to Traditional Key Derivation

Scheme	Output Size	Hardness Dependency	Quantum Resistance
PBKDF2	Limited	Memory-hard / Iterative	Moderate
Argon2	Limited	Memory-hard / Iterative	Moderate
This Scheme	12,800-bit	Nonlinear diffusion only	High

The design avoids computationally expensive iterations while achieving robust quantum resistance.

---

6. Hash-Based Post-Quantum Signatures

The derived coordinates can act as a deterministic seed for a **hash-based signature scheme**:

- **Public/Private Key Asymmetry:** Uses only hash functions.
- **Quantum Security:** Resistant to both Shor and Grover attacks.
- **Simplicity:** Minimalistic, audit-friendly, and suitable for constrained devices.
- **Potential Schemes:** SPHINCS+, XMSS, or custom simplified hash-based signatures.

Workflow:

1. Derive 12,800-bit coordinate set from passphrase.
2. Use a subset or full output as deterministic seed for the signature private key.
3. Generate public key via hash-based construction.
4. Sign messages using hash-based scheme; verify with public key.

This approach combines usability (human-memorable passphrases) with post-quantum robustness.

---

7. Applications

- Quantum-safe seed generation from passphrases
- Secure brainwallets resistant to quantum attacks
- Generation of cryptographic lattices for zero-knowledge proofs
- Decentralized identity and key derivation
- Deterministic input for hash-based post-quantum signatures

---

8. Implementation Example (Rust)

```rust
use blake3::Hasher;

// Helper function: rotate
fn rotate_left_u128(value: u128, n: u32) -> u128 {
    value.rotate_left(n)
}

// Mixing function with multiple rounds
fn mix(val1: u128, val2: u128, round: u32) -> u128 {
    let mut v = val1 ^ val2 ^ (round as u128);
    for i in 0..8 {
        let rot_bits = ((i * 7 + round * 3) % 128) as u32;
        v = v.rotate_left(rot_bits);
        v = v.wrapping_mul(0x9E3779B97F4A7C15);
        v ^= v >> 23;
        v = v.wrapping_add(round as u128);
    }
    v
}

// Derive 100 coordinates from passphrase + optional salt
fn derive_coordinates_per_word(keyphrase: &str, salt: &str) -> Vec<u128> {
    let words: Vec<&str> = keyphrase.split_whitespace().collect();
    let word_count = words.len();
    let base_coords_per_word = 100 / word_count;
    let remainder = 100 % word_count;

    let mut coords = Vec::with_capacity(100);

    for (idx, word) in words.iter().enumerate() {
        let coords_for_word = if idx < remainder { base_coords_per_word + 1 }
else { base_coords_per_word };
        let mut hasher = Hasher::new();
```

```
        hasher.update(salt.as_bytes());
        hasher.update(word.as_bytes());
        let hash = hasher.finalize();
        let hash_bytes = hash.as_bytes();
        let base_val1 =
u128::from_be_bytes(hash_bytes[0..16].try_into().unwrap());
        let base_val2 =
u128::from_be_bytes(hash_bytes[16..32].try_into().unwrap());

        for i in 0..coords_for_word {
            let mixed_val = mix(
                rotate_left_u128(base_val1, (i % 128) as u32),
                rotate_left_u128(base_val2, ((i * 3) % 128) as u32),
                i as u32,
            );
            coords.push(mixed_val);
        }
    }

    coords.truncate(100);
    coords
}

fn main() {
    let keyphrase = "silver ladder dove cloud whisper apple horse river matrix
zero jazz velvet canyon galaxy timber foam vivid spring echo marble giant zero
rapid";
    let salt = "optional_salt";

    println!("Using keyphrase: \"{}\"", keyphrase);
    println!("Using salt: \"{}\"\n", salt);

    let coords = derive_coordinates_per_word(keyphrase, salt);

    // 1. Single Coordinates
    for (i, c) in coords.iter().enumerate() {
        println!("Coordinate {:03}: {:032x}", i + 1, c);
    }

    // 2. Lattice (Decimal)
    let lattice_dec = coords.iter()
        .map(|c| c.to_string())
        .collect::<Vec<String>>()
        .join(",");
    println!("\nLattice grid (decimal):\n{}", lattice_dec);

    // 3. Lattice (Hex)
    let lattice_hex = coords.iter()
        .map(|c| format!("{:032x}", c))
        .collect::<Vec<String>>()
```

```
        .join(",");
    println!("\nLattice grid (hex):\n{}", lattice_hex);

    // 4. 10x10
    println!("\n--- 10x10 Gitter (Hex) ---");
    for row in 0..10 {
        for col in 0..10 {
            let idx = row * 10 + col;
            print!("{:032x} ", coords[idx]);
        }
        println!();
    }

    let total_bits = coords.len() * 128;
    println!("\n=== Quantum Security Debug ===");
    println!("Total output bits: {} bits", total_bits);
    println!("Classical brute-force complexity: 2^{}", total_bits);
    println!("Grover's algorithm reduces this to ~2^{}", total_bits / 2);
    println!("Shor's algorithm does NOT apply (no algebraic structure)");
}
```

Disclaimer: This software is provided 'as-is', without any express or implied warranty. Use at your own risk. The author is not responsible for any damages resulting from its use.

9. Security Analysis

- **Classical Brute Force:** 2^12800 operations
- **Grover's Attack:** ~2^6400 operations
- **Shor's Algorithm:** Not applicable
- **Memory-Hardness:** Not required, low computational cost
- **Preimage Attacks:** Practically infeasible due to extreme output width
-

---

10. Conclusion

This proposal presents a quantum-resistant, deterministic coordinate derivation scheme from natural language passphrases, suitable as a seed for post-quantum hash-based signatures. By combining large output size with nonlinear mixing, it offers robust security against known quantum attacks, simplicity of implementation, and versatility for a wide range of applications. This method aligns with ICCS goals for next-generation public-key cryptography: secure, auditable, and post-quantum ready.

Personal Note

The wheel does not need to be reinvented – the goal is to optimize and extend existing systems. With our approach, this is exactly what becomes possible: current security mechanisms can be upgraded quickly and efficiently.

The use of **BLAKE3** here serves as a fast example; in principle, any modern hash function could be used, or even a new one developed. Similarly, the **mixing function** presented is a simple demonstration that can be elaborated and refined.

The **100 coordinates** provide very high security but come at a computational cost. They could be reduced for efficiency, or in highly sensitive scenarios, even increased to **1,000 coordinates**.

My recommendation for passphrase entropy: At least 12–20 random words (BIP39-like) or a combination with salt.

The essence of our idea is to extend existing systems in a simple, fast, and secure way against quantum attacks – making the search space effectively unbreakable for adversaries. Instead of the hash, tasks could be integrated that quantum computers cannot efficiently solve. Likewise, the mixing function could be further modified. Additional validations, or even a **minimum difficulty per coordinate**, could be introduced, with a random salt that must be computed—similar to a nonce in crypto-mining.

Overall, this system is highly **extensible and secure**, providing a foundation for innovation in post-quantum cryptography.

---

11. Code Debug / Test

Running `target\debug\my_coords.exe`

Using keyphrase: "silver ladder dove cloud whisper apple horse river matrix zero jazz velvet canyon galaxy timber foam vivid spring echo marble giant zero rapid"

Using salt: "optional_salt"

Coordinate 001: 310be03c175297e05b9998ded8c901c5

Coordinate 002: f1d94c395a14ae974ec2815e739e14dd

Coordinate 003: f3b8910848b3aeb29e822ea29f04a362

Coordinate 004: 5a4a972e35d74fc00903c39e0349f058

Coordinate 005: 77a3f69fe386404886ef85187ef3abb1

Coordinate 006: ce3552105d4c4e2c1165fc75f559dc66

Coordinate 007: 797c3b69631f5cd048741e0ef86e6181

Coordinate 008: 5a42e43f796c86bdd2d94853541401b2

Coordinate 009: 9930cbf910a232674dc2e8317b1a6fb3

Coordinate 010: dedda7320baa5c3d532db2f34035d018

Coordinate 011: 53f18ab5eb07604d8a449f04f537c8b3

Coordinate 012: 5c5311a14a16063125f82864f9fc9b25

Coordinate 013: 5defd1dd6bf82fe011ee84a963e19871

Coordinate 014: c1a3816eae7a43cfe3756c2dc7b343c7

Coordinate 015: 43a7efa73056367dcae2d4d17d47f26f

Coordinate 016: 741051728f6f2f855bd98be77eb036d6

Coordinate 017: 8d5481783f2dd2a4d3dd8adca1d09f85

Coordinate 018: 01b59a416ec3eeed209d20a0af665250

Coordinate 019: 70af05a3cf95d38970ab68ff480ba65c

Coordinate 020: cfc621e3ca02e302eaa6abec55b8ab4b

Coordinate 021: 85e6cb3a1fe5a2700e6f3d4d153de4b1

Coordinate 022: c74912f848fc61f811a754fd30fa016c

Coordinate 023: dc13ed59f164d6396111920fde8c974e

Coordinate 024: fce9acce299f978456843d9620de4c7b

Coordinate 025: 9b5362169a62011138c213aa63548d9e

Coordinate 026: ddf56c658d9ff8fc5375061a9804ab0b

Coordinate 027: 81b4274e37a588333be9789f9c0a19c6

Coordinate 028: bc31fbc7e49166c663c9a5509388c26f

Coordinate 029: d817a9b765e9096bd47a8b7c2208e703

Coordinate 030: 3e60c8806efde516be9b93f268772cd3

Coordinate 031: cc2e95bfe252815ffb74b70b52359fd0

Coordinate 032: e24d909d60b6d968a52fe440432ebf43

Coordinate 033: b869b9fe248e0f9eb72e78d2cfea6820

Coordinate 034: d912c8194edfdc9611ae011a840eae95

Coordinate 035: 67389ce62388460a3a89759acace91b5

Coordinate 036: 8b4b6b00f2e8283815291c1244108b4e

Coordinate 037: dd6d185de19087493b2e027847fd0efe

Coordinate 038: 9eea6bedbec58d450b0eaa16ad69c4ca

Coordinate 039: 3e09f8611105f4703c70ddbcbbd726e8

Coordinate 040: 2107eda05655f3529a794619e21e20b5

Coordinate 041: 5dc061bff1220aff269313e524f580d4

Coordinate 042: aac7e892a1e77e4cd329a7bb3b3ed6a6

Coordinate 043: 9b416ec3288dffd83b61885331b4e3b5

Coordinate 044: 2635c656e8140bddc7b9a1ec4778dcc5

Coordinate 045: 818740d11356efa816474f0fc0ec80e6

Coordinate 046: 9a20122da3f0d40e24bf39597dc0000e

Coordinate 047: e84688a096ccca4dde7f5f8cf5011a5f

Coordinate 048: 545f4084c24febb4b0b77b478c1b0769

Coordinate 049: 40d8da685081499adb64e8c2eb5459f1

Coordinate 050: b2ca0bdf25c87900a4d99edd8348e7f4

Coordinate 051: 86a10c4045287b0a84f29a83882c403e

Coordinate 052: 36a22c0a87be8debf15fe84876c0e8fe

Coordinate 053: d27485a40f269a6fbc350c9c818a94ef

Coordinate 054: 049ce01e991189e5a706b23d6cd7a622

Coordinate 055: 2cd6b4a5717c8b885c00bc3bc2f0aa7d

Coordinate 056: ad33083a40293138c575e893977a1f97

Coordinate 057: 26d36f55347297171aa973671a63f7a2

Coordinate 058: ea8700402d8640ae6c0954b532323834

Coordinate 059: 598b1ae759f06c7189dfea99a19478e3

Coordinate 060: a4ffc50014ac8df428c85ad1b0a0c5b6

Coordinate 061: 8d54960d5aa045638fe19d6967370ec8

Coordinate 062: 4eba0cf6d32cdf83ef788010cfc13cc3

Coordinate 063: 93c16e07b9e8947e5fe34a27c2c5ae52

Coordinate 064: f70e31699f22a5bedf47e824280a9006

Coordinate 065: 882ffc08c93a59b40a638b92ad84cb6a

Coordinate 066: 685234aea1358f792345c378509e9a31

Coordinate 067: bb8a7c3f771789c5b96d89931be06b15

Coordinate 068: 24185a5dc70e8a5ed796a18a4844b512

Coordinate 069: 831451182478c6af694b882a8af5d344

Coordinate 070: c22288f3137c14b31a817f69c1bfc32c

Coordinate 071: d4c323d899933e43dcc70c737ef9b39e

Coordinate 072: 9acd801b5e1fbc5d73f407125707dbdb

Coordinate 073: ce5ca035008fc981dbd8cfbdc72fa127

Coordinate 074: bede2d5853bf096fe6574ae823133792

Coordinate 075: 81ebc029eee7e50f99e385d938a69956

Coordinate 076: e9b9c726c85712e8dfd15acfcff09daa

Coordinate 077: fdbf81911b5c1ebbe2bd4dffdd96c4fe

Coordinate 078: ec72b80904d36b0a2a6ffb68d9daeae0

Coordinate 079: 719a42308605b285fee9be86cf732b80

Coordinate 080: 7c90021e11d223e8691c8d6a81492625

Coordinate 081: 15de63a76f87ad9eeac8c41ebc480aeb

Coordinate 082: 2b931230f2ed7ad43cb4fe5d128395c4

Coordinate 083: 9015106a6430b5d27a3fb6572b65cf1c

Coordinate 084: 74b448094cc4b3fd5944c9cf57a85930

Coordinate 085: 20f52cde84ca77c752a4164da6bb18c1

Coordinate 086: 8d2233ae130081c46e61995f029f9b00

Coordinate 087: 2494ad1d666345b45e78eafd28b1724f

Coordinate 088: 8c6a2be4011fe293981e6ed62aea1855

Coordinate 089: b010ea51d57f62c38e53c6acb0050342

Coordinate 090: b26c21292d0bc8166b87ce61d7af7b32

Coordinate 091: 984ba8c5ec9f6c0996b3fb5a2e61e65d

Coordinate 092: e837f4b9c7398f1f90a9327b356710b1

Coordinate 093: 818740d11356efa816474f0fc0ec80e6

Coordinate 094: 9a20122da3f0d40e24bf39597dc0000e

Coordinate 095: e84688a096ccca4dde7f5f8cf5011a5f

Coordinate 096: 545f4084c24febb4b0b77b478c1b0769

Coordinate 097: 84128648c8af6d51d14a166c2eb5c328

Coordinate 098: 12716c1c04e57143912cd6c8cf4b9821

Coordinate 099: 4e1a165096eb812dadd70cb69b80ec8c

Coordinate 100: 2473c455b7ca527932a7e542d54a4c4f

Lattice grid (decimal):

651938350795637340653449980224214798 77,3214722214094805776884430459943475007
65,3239607272034229384211589709624647770 58,1200178158931826340453724581154 39
128664,1590294780268019007165747838074174616 81,2740978233193074687808724278 33
3587784806,1614816373122337496661901336134768643 85,1199778406316484846399732
98620293448114,2036252506664116517852292364170120682 75,2962395037974097443 19
058824595395432472,1115800805786155244881537506061587437 31,1227202938310965 9
421091023535 9864979237,1248634191236330145735458921567927 39953,2573899727743

64888900774585625316901831,899302500349225977500593543596227917911542751762
12365992492026632621660321494,18785992629956138839555989696032542707722721 6
24023694653707192075015090550561497823018686372066184499058582421274522761
78957272296754671210147325171772235,17798567365111143497479512068026098603 3,
26489579358641650305165686447918979518029253362677008397457298805709469939 0
798,33617876501417254096151062843440590348320646328945286025545363545882695
4378654,29503369834491983039344457373819788775517240582210178342659284459131
24002392518,25015439247558560939173114977160585278328723611217480459702369 0
678641237747459,82914662894559950886493106478100458707,27140439407726877955 9
33936240650706120030080826704123422170960943322229314336324512691477561383
32807025071201867796802885399949158156274208836873064776126931372044344792
96553207227557708705993141,18515428397149704223062219528663784123029432584 1
64190515698810375581538746751821123520985285694157307685560066203770682463
90413837351119047201810186549015243905689573213653925532112718298751157,124
61710720577350029642672886845784085222700674349473084824276921725680083523 8
,20637008516956588318688520969885823070950787983758774051242315717006102929
33,17217268617104226307593658081984410442220486763354972753798344552668893
8590222,30874712693308034705201666078309096508717214972843996253767279157387
77099267945,86196557681615735987048515307142666737,23765166800121824335269854
8453691804660,17895275971032388270584836719994026399872620357123783127466 4
29732721583515902,27974289610900725496028101632454836759961314559770395994 4
59183709219124444502596008472837605955103370756118681135332302214172849428 1
62924827959980946963435160849657506922240553158852631094672231174031617426
40438530221825497341768201190235665603839096481019437097599940832193214226
48742809375592106869599028662,18786034376124413073117545704410834912810464 5
81383130972368434744795134225529919640086035126929318130111847065909205032
83930093211713538745468858265298903101810241572424221442319277403783994274 3
4,13866654842492261119907308867717317278529866548424922611199073088677173172785,24928469222527933772570438886412729 8
98325,47978655819550345116568971538562594066,17423435817293594174687108357323 6
36323140,25804954694164367081556486334646424452428280956003896489996944393 6
681936401310,20576813052359629654240219860137829065127430190782748089946662
6358480550732071,25370692880815741837789068769419927336217269449876294617221 4
43807144468534132063106747372088979115312689557504713435303373289039560953
76513298094270628120294231429346152502717975865110136347812937615100371972
33733777860377262517085254401655720051721379553089789906421049687412906849
90382487274387524326489624440115792044041846313645082275037580594938019151
82025748033900695285490142825551641551265220158900959627525357922612902884
38083186512297694925415424264426682251875987336933342139313906869485185011 2
0,48624178969485754394636446495327482447,18664319308487495589694164846742300
8853,23403195657528447608974081164807117293023716402389224943886357040309578
1276466,20243550074968994525984049778783799868530867143497613035578658405 0
718827024561,17217268617104226307593658081984410442220486763354972753798344
5526688938590222,30874712693308034705201666078309096508711214972843996253767
27915738770992679451755542803964861927014005641261994319762451502618928398
49398822318494079775051038152359875170841514525444623932407164845330413055
44955324315919358345497679

Lattice grid (hex):

310be03c175297e05b9998ded8c901c5,f1d94c395a14ae974ec2815e739e14dd,f3b8910848b3
aeb29e822ea29f04a362,5a4a972e35d74fc00903c39e0349f058,77a3f69fe386404886ef85187e
f3abb1,ce3552105d4c4e2c1165fc75f559dc66,797c3b69631f5cd048741e0ef86e6181,5a42e43
f796c86bdd2d94853541401b2,9930cbf910a232674dc2e8317b1a6fb3,dedda7320baa5c3d532
db2f34035d018,53f18ab5eb07604d8a449f04f537c8b3,5c5311a14a16063125f82864f9fc9b25,
5defd1dd6bf82fe011ee84a963e19871,c1a3816eae7a43cfe3756c2dc7b343c7,43a7efa7305636
7dcae2d4d17d47f26f,741051728f6f2f855bd98be77eb036d6,8d5481783f2dd2a4d3dd8adca1d
09f85,01b59a416ec3eeed209d20a0af665250,70af05a3cf95d38970ab68ff480ba65c,cfc621e3c
a02e302eaa6abec55b8ab4b,85e6cb3a1fe5a2700e6f3d4d153de4b1,c74912f848fc61f811a754f
d30fa016c,dc13ed59f164d6396111920fde8c974e,fce9acce299f978456843d9620de4c7b,9b53
62169a62011138c213aa63548d9e,ddf56c658d9ff8fc5375061a9804ab0b,81b4274e37a588333
be9789f9c0a19c6,bc31fbc7e49166c663c9a5509388c26f,d817a9b765e9096bd47a8b7c2208e7
03,3e60c8806efde516be9b93f268772cd3,cc2e95bfe252815ffb74b70b52359fd0,e24d909d60b
6d968a52fe440432ebf43,b869b9fe248e0f9eb72e78d2cfea6820,d912c8194edfdc9611ae011a8
40eae95,67389ce62388460a3a89759acace91b5,8b4b6b00f2e8283815291c1244108b4e,dd6d1
85de19087493b2e027847fd0efe,9eea6bedbec58d450b0eaa16ad69c4ca,3e09f8611105f4703c7
0ddbcbbd726e8,2107eda05655f3529a794619e21e20b5,5dc061bff1220aff269313e524f580d4,
aac7e892a1e77e4cd329a7bb3b3ed6a6,9b416ec3288dffd83b61885331b4e3b5,2635c656e8140
bddc7b9a1ec4778dcc5,818740d11356efa816474f0fc0ec80e6,9a20122da3f0d40e24bf39597dc
0000e,e84688a096ccca4dde7f5f8cf5011a5f,545f4084c24febb4b0b77b478c1b0769,40d8da68
5081499adb64e8c2eb5459f1,b2ca0bdf25c87900a4d99edd8348e7f4,86a10c4045287b0a84f29
a83882c403e,36a22c0a87be8debf15fe84876c0e8fe,d27485a40f269a6fbc350c9c818a94ef,049
ce01e991189e5a706b23d6cd7a622,2cd6b4a5717c8b885c00bc3bc2f0aa7d,ad33083a4029313
8c575e893977a1f97,26d36f55347297171aa973671a63f7a2,ea8700402d8640ae6c0954b5323
23834,598b1ae759f06c7189dfea99a19478e3,a4ffc50014ac8df428c85ad1b0a0c5b6,8d54960d
5aa045638fe19d6967370ec8,4eba0cf6d32cdf83ef788010cfc13cc3,93c16e07b9e8947e5fe34a
27c2c5ae52,f70e31699f22a5bedf47e824280a9006,882ffc08c93a59b40a638b92ad84cb6a,685
234aea1358f792345c378509e9a31,bb8a7c3f771789c5b96d89931be06b15,24185a5dc70e8a5
ed796a18a4844b512,831451182478c6af694b882a8af5d344,c22288f3137c14b31a817f69c1bf
c32c,d4c323d899933e43dcc70c737ef9b39e,9acd801b5e1fbc5d73f407125707dbdb,ce5ca0350
08fc981dbd8cfbdc72fa127,bede2d5853bf096fe6574ae823133792,81ebc029eee7e50f99e385d
938a69956,e9b9c726c85712e8dfd15acfcff09daa,fdbf81911b5c1ebbe2bd4dffdd96c4fe,ec72b8
0904d36b0a2a6ffb68d9daeae0,719a42308605b285fee9be86cf732b80,7c90021e11d223e8691
c8d6a81492625,15de63a76f87ad9eeac8c41ebc480aeb,2b931230f2ed7ad43cb4fe5d128395c4,
9015106a6430b5d27a3fb6572b65cf1c,74b448094cc4b3fd5944c9cf57a85930,20f52cde84ca7
7c752a4164da6bb18c1,8d2233ae130081c46e61995f029f9b00,2494ad1d666345b45e78eafd2
8b1724f,8c6a2be4011fe293981e6ed62aea1855,b010ea51d57f62c38e53c6acb0050342,b26c21
292d0bc8166b87ce61d7af7b32,984ba8c5ec9f6c0996b3fb5a2e61e65d,e837f4b9c7398f1f90a9
327b356710b1,818740d11356efa816474f0fc0ec80e6,9a20122da3f0d40e24bf39597dc0000e,e
84688a096ccca4dde7f5f8cf5011a5f,545f4084c24febb4b0b77b478c1b0769,84128648c8af6d5
1d14a166c2eb5c328,12716c1c04e57143912cd6c8cf4b9821,4e1a165096eb812dadd70cb69b8
0ec8c,2473c455b7ca527932a7e542d54a4c4f

--- 10x10 Gitter (Hex) ---

310be03c175297e05b9998ded8c901c5 f1d94c395a14ae974ec2815e739e14dd
f3b8910848b3aeb29e822ea29f04a362 5a4a972e35d74fc00903c39e0349f058
77a3f69fe386404886ef85187ef3abb1 ce3552105d4c4e2c1165fc75f559dc66

797c3b69631f5cd048741e0ef86e6181 5a42e43f796c86bdd2d94853541401b2
9930cbf910a232674dc2e8317b1a6fb3 dedda7320baa5c3d532db2f34035d018

53f18ab5eb07604d8a449f04f537c8b3 5c5311a14a16063125f82864f9fc9b25
5defd1dd6bf82fe011ee84a963e19871 c1a3816eae7a43cfe3756c2dc7b343c7
43a7efa73056367dcae2d4d17d47f26f 741051728f6f2f855bd98be77eb036d6
8d5481783f2dd2a4d3dd8adca1d09f85 01b59a416ec3eeed209d20a0af665250
70af05a3cf95d38970ab68ff480ba65c cfc621e3ca02e302eaa6abec55b8ab4b

85e6cb3a1fe5a2700e6f3d4d153de4b1 c74912f848fc61f811a754fd30fa016c
dc13ed59f164d6396111920fde8c974e fce9acce299f978456843d9620de4c7b
9b5362169a62011138c213aa63548d9e ddf56c658d9ff8fc5375061a9804ab0b
81b4274e37a588333be9789f9c0a19c6 bc31fbc7e49166c663c9a5509388c26f
d817a9b765e9096bd47a8b7c2208e703 3e60c8806efde516be9b93f268772cd3

cc2e95bfe252815ffb74b70b52359fd0 e24d909d60b6d968a52fe440432ebf43
b869b9fe248e0f9eb72e78d2cfea6820 d912c8194edfdc9611ae011a840eae95
67389ce62388460a3a89759acace91b5 8b4b6b00f2e8283815291c1244108b4e
dd6d185de19087493b2e027847fd0efe 9eea6bedbec58d450b0eaa16ad69c4ca
3e09f8611105f4703c70ddbcbbd726e8 2107eda05655f3529a794619e21e20b5

5dc061bff1220aff269313e524f580d4 aac7e892a1e77e4cd329a7bb3b3ed6a6
9b416ec3288dffd83b61885331b4e3b5 2635c656e8140bddc7b9a1ec4778dcc5
818740d11356efa816474f0fc0ec80e6 9a20122da3f0d40e24bf39597dc0000e
e84688a096ccca4dde7f5f8cf5011a5f 545f4084c24febb4b0b77b478c1b0769
40d8da685081499adb64e8c2eb5459f1 b2ca0bdf25c87900a4d99edd8348e7f4

86a10c4045287b0a84f29a83882c403e 36a22c0a87be8debf15fe84876c0e8fe
d27485a40f269a6fbc350c9c818a94ef 049ce01e991189e5a706b23d6cd7a622
2cd6b4a5717c8b885c00bc3bc2f0aa7d ad33083a40293138c575e893977a1f97
26d36f55347297171aa973671a63f7a2 ea8700402d8640ae6c0954b532323834
598b1ae759f06c7189dfea99a19478e3 a4ffc50014ac8df428c85ad1b0a0c5b6

8d54960d5aa045638fe19d6967370ec8 4eba0cf6d32cdf83ef788010cfc13cc3
93c16e07b9e8947e5fe34a27c2c5ae52 f70e31699f22a5bedf47e824280a9006
882ffc08c93a59b40a638b92ad84cb6a 685234aea1358f792345c378509e9a31
bb8a7c3f771789c5b96d89931be06b15 24185a5dc70e8a5ed796a18a4844b512
831451182478c6af694b882a8af5d344 c22288f3137c14b31a817f69c1bfc32c

d4c323d899933e43dcc70c737ef9b39e 9acd801b5e1fbc5d73f407125707dbdb
ce5ca035008fc981dbd8cfbdc72fa127 bede2d5853bf096fe6574ae823133792
81ebc029eee7e50f99e385d938a69956 e9b9c726c85712e8dfd15acfcff09daa
fdbf81911b5c1ebbe2bd4dffdd96c4fe ec72b80904d36b0a2a6ffb68d9daeae0
719a42308605b285fee9be86cf732b80 7c90021e11d223e8691c8d6a81492625

15de63a76f87ad9eeac8c41ebc480aeb 2b931230f2ed7ad43cb4fe5d128395c4
9015106a6430b5d27a3fb6572b65cf1c 74b448094cc4b3fd5944c9cf57a85930
20f52cde84ca77c752a4164da6bb18c1 8d2233ae130081c46e61995f029f9b00
2494ad1d666345b45e78eafd28b1724f 8c6a2be4011fe293981e6ed62aea1855
b010ea51d57f62c38e53c6acb0050342 b26c21292d0bc8166b87ce61d7af7b32

984ba8c5ec9f6c0996b3fb5a2e61e65d e837f4b9c7398f1f90a9327b356710b1
818740d11356efa816474f0fc0ec80e6 9a20122da3f0d40e24bf39597dc0000e
e84688a096ccca4dde7f5f8cf5011a5f 545f4084c24febb4b0b77b478c1b0769
84128648c8af6d51d14a166c2eb5c328 12716c1c04e57143912cd6c8cf4b9821
4e1a165096eb812dadd70cb69b80ec8c 2473c455b7ca527932a7e542d54a4c4f

=== Quantum Security Debug ===

Total output bits: 12800 bits

Classical brute-force complexity: 2^12800

Grover's algorithm reduces this to ~2^6400

Shor's algorithm does NOT apply (no algebraic structure)