Poland

Global Quantum HACKATHON

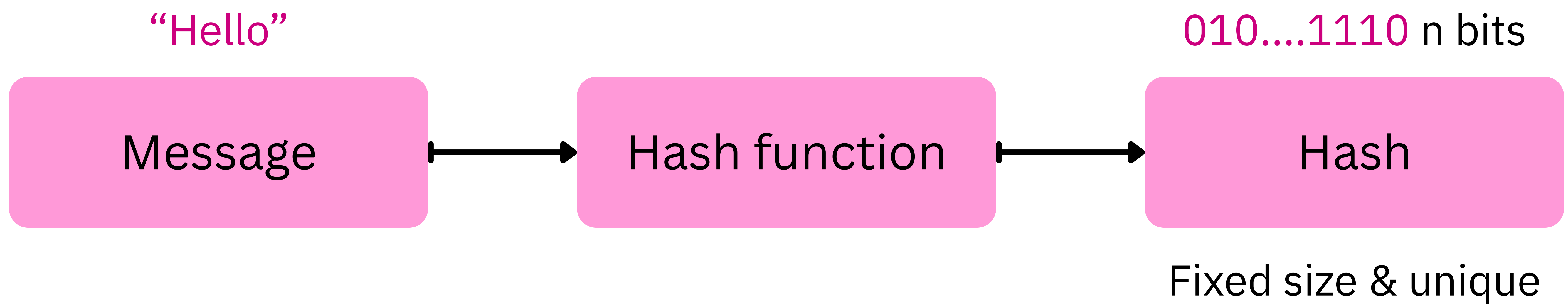
Quantum Graph Hash (QGH_256)

Using Classical Random Walks + Spectral Fingerprinting using Quantum Phase Estimation Algorithms

Open-track

Presented by ~ Quantum Walkers
Mohana Priya Thinesh Kumar
Pranavishvar Hariprakash
IIT(ISM) Dhandbad

What is Hash Function?



Purpose:
A hash function is a mathematical function that takes a string of bits as input ('message') and returns a fixed-size string of bytes ('hash')

Properties:
Fast to compute and ideally unique for each input.

Properties of Hash Function

1. **Deterministic generation:**
The same input always produces the same hash output.

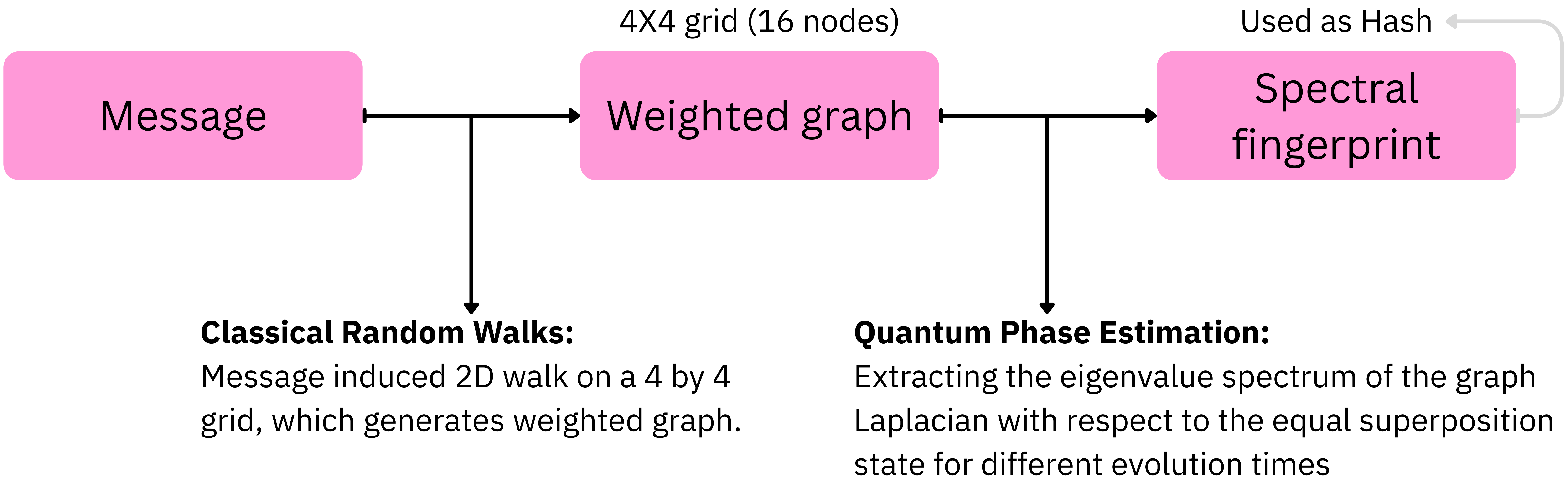
2. **Fast computation:**
The hash value is computed quickly for any given input.

3. **Pre-image resistance:**
It is computationally infeasible to find the input from a given hash.

4. **Collision resistance:**
It is difficult to find two different inputs that produce the same hash output.

5. **Avalanche Effect:**
A small change in the input causes a drastically different output

Workflow of Quantum Graph Hash



Graph Generation from the Message

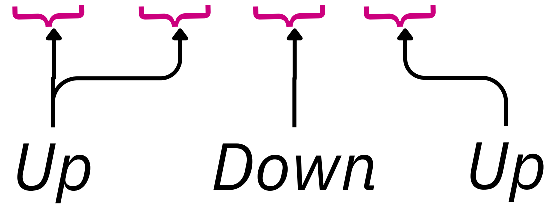
Steps to generate weighted graph from the message

- Convert message into UTF-8 binary form.
- Split the bits into 2-bit blocks (add a 0 if odd).
- Each block controls walker’s move:
    a. *00 → Down*
    b. *01 → Up*
    c. *10 → Right*
    d. *11 → Left*
- The walker performs a 2D walk based on the message.
- Each time an edge is revisited, its weight increases.

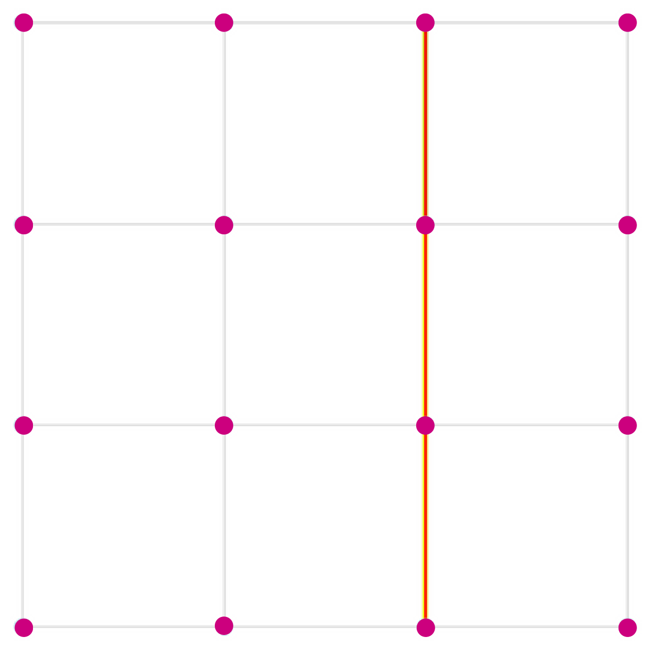
Example

Message: “Q”

UTF-8: 01 01 00 01



Graph:



Spectral Fingerprinting

Steps to extract spectral fingerprint of a graph

- Compute the graph Laplacian (L = Degree matrix - Adjacency matrix).
- Trotterize the Laplacian using the Trotter–Suzuki decomposition.
- Convert the Trotterized form into a unitary circuit e^{-iLt}
- Simulate this unitary for different evolution times t.
- Use Quantum Phase Estimation (QPE) to extract the Laplacian eigenvalues with respect to the equal superposition state.
- Compute the heat trace $h(t)=\sum p_i e^{-t\lambda_i}$ for each t.
- Collect all h(t) values into a vector (spectral fingerprint)

Spectral fingerprint

uniquely identifies a graph by capturing its structural features

Laplacian

- Compute the graph Laplacian (L = Degree matrix - Adjacency matrix).
- It explains the overall connectivity of the graph

Graph Laplacian for "Hello"

```
[[ 4.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  4.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  4. -2.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -2.  4.  0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0. -1.]
 [ 0.  0.  0.  0.  4. -1.  0. -1.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0. -1.  4. -5.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0. -1.  0.  0. -5.  4.  0.  0.  0. -6.  0.  0.  0.  0.  0.]
 [ 0.  0.  0. -1. -1.  0.  0.  4.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  4.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  4.  0.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0. -6.  0.  0.  0.  4. -1.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0. -1.  4.  0.  0.  0. -1.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  4.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  4.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  4.  0.]
 [ 0.  0.  0. -1.  0.  0.  0.  0.  0.  0.  0. -1.  0.  0.  0.  4.]]
```

```python
# Out-degree Laplacian for directed graph
degrees = dict(G.degree())  # use out-degree for directed case
D = np.diag([degrees[node] for node in G.nodes()])
L = D - A  # directed Laplacian (non-Hermitian)
print(L)
```

Matrix Exponentiation and Trotter-Decomposition

- The Laplacian we get is a Hermitian matrix, we need a Unitary matrix for our further processing, for this conversion we use the Matrix Exponentiation and Trotter-Decomposition
- Matrix exponentiation: We convert a Hermitian into a Unitary by raising the matrix to its exponent.

$$U = e^{iLt}$$

$$U^\dagger = \left(e^{iLt}\right)^\dagger = e^{-iLt}$$

$$U^\dagger U = e^{-iLt} e^{iLt} = e^0 = I$$

- Trotter Decomposition: It is a technique for approximating the exponential of a sum of non-commuting operators (AB != BA)

$$e^{i(A+B)t} \neq e^{iAt} e^{iBt}.$$

$$e^{i(A+B)t} \approx \left(e^{iAt/r}\, e^{iBt/r}\right)^r$$

```
trotter_circuits = {}
unitary_gates = {}
for t in times:
    evo_gate = PauliEvolutionGate(pauli_op, time=t, synthesis=SuzukiTrotter(order=2, reps=3))
    trotter_circuits[t] = evo_gate
    unitary_gates[t] = UnitaryGate(Operator(evo_gate).data, label=f"U({t})")
```

Quantum Phase Estimation

- QPE finds the phase (angle) associated with how a unitary operator U acts on its eigenvector
- These angles give us data about the eigen values which reveal further information about the graph.

$$U|k_i\rangle = e^{i\alpha_i}|k_i\rangle$$

$$L|k_i\rangle = \lambda_i|k_i\rangle$$

$$U|k_i\rangle = e^{iLt}|k_i\rangle = e^{i\lambda_i t}|k_i\rangle$$

- Phase: $\alpha_i = \lambda_i t$
- Eigen Values of U: $e^{i\alpha_i} = e^{i\lambda_i t}$

```
superposition_state = np.zeros(2**n_qubits)
for i in range(n):
  superposition_state[i] = 1
superposition_state /= np.linalg.norm(superposition_state)

spectrum = {}
for t, unitary in unitary_gates.items():
  num_ancilla = 3
  initial_state = QuantumCircuit(n_qubits)
  initial_state.initialize(superposition_state)

  phase_estimation = PhaseEstimation(num_evaluation_qubits=num_ancilla, sampler=Sampler())
  simulator = Aer.get_backend("aer_simulator_statevector")
  circ = phase_estimation.construct_circuit(unitary, initial_state)
  circ.measure_all()
  transpiled = transpile(circ, simulator)
  job = simulator.run(transpiled, shots = 300, seed_simulator = 42)
  result = job.result()
  counts = result.get_counts()
  spectrum[t] = counts

spectrum_collection.append(spectrum)
```

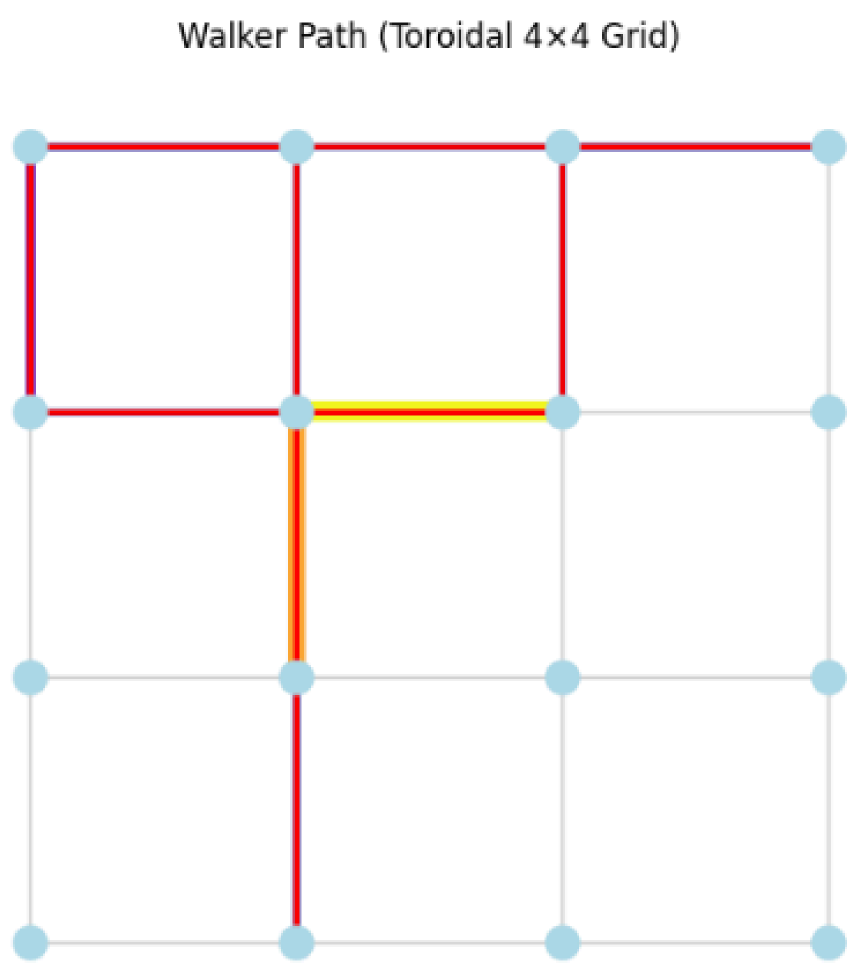
Intuition Behind Spectral Fingerprinting

- The graph is treated as a dissipative system, with each node holding initial heat defined by the graph Laplacian.
- An equal superposition state (all nodes equally excited) is evolved under the Laplacian for time t.
- Quantum Phase Estimation (QPE) extracts the eigenvalue spectrum of the Laplacian with respect to the equal superposition state.
- The heat trace $h(t)=\sum e^{-t\lambda_i}$ represents the total heat remaining after time t.
- Repeating this for multiple t values forms a spectral fingerprint vector, uniquely capturing the graph's heat dissipation pattern and structure.

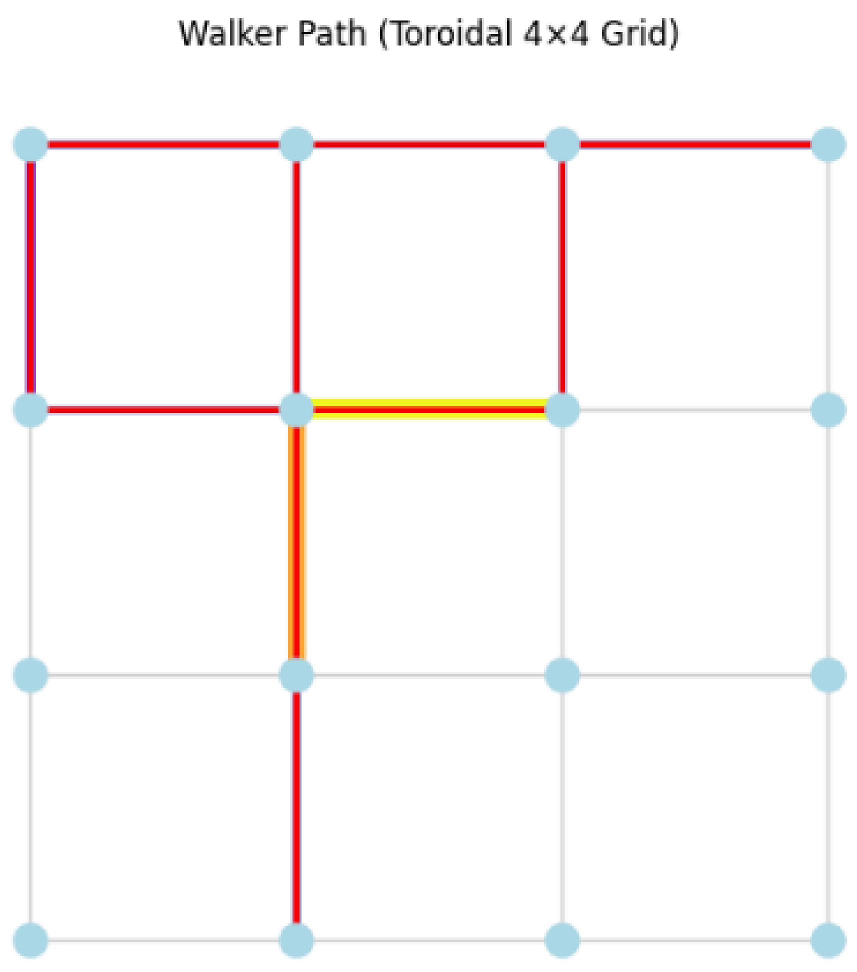
Advantages of our Hash Function

Deterministic generation: The walker's moves are fully determined by the user's UTF-8 message blocks, so the same input always produces the same weighted graph and adjacency matrix.

```
sf1 = spectral_fingerprint("hello")
hash1 = convert_spectral_fingerprint_into_hash(sf1)
print(hash1)
```




```
sf2 = spectral_fingerprint("hello")
hash2 = convert_spectral_fingerprint_into_hash(sf2)
print(hash2)
```

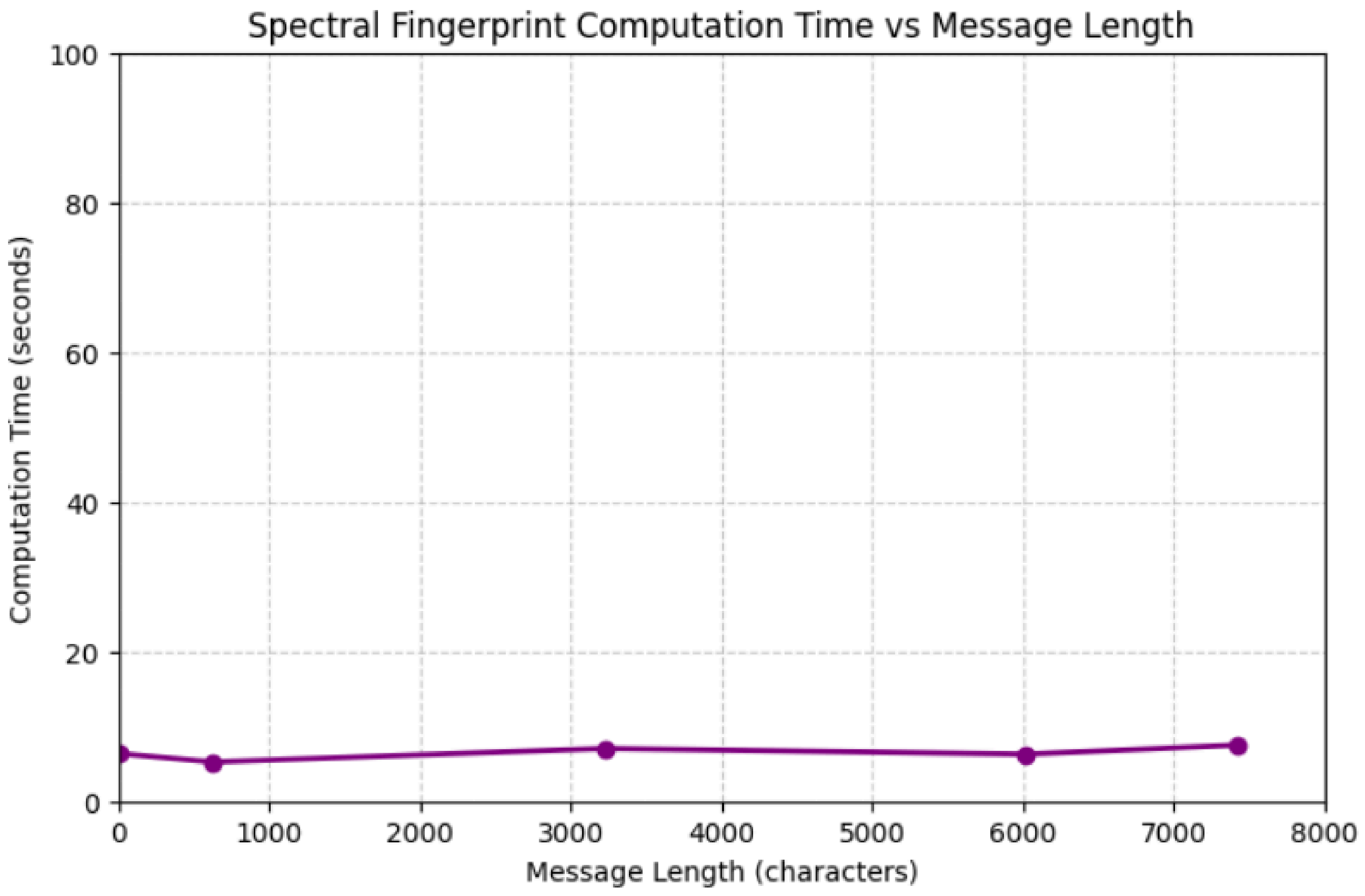



```
# TESTING
def hamming_distance(a, b):
    return sum(x != y for x, y in zip(a, b))

# Test 1: Collision resistance
print(f"Collision: {hash1 == hash2}")

# Test 2: Avalanche effect
dist = hamming_distance(hash1, hash2)
print(f"Hamming distance (avalanche): {dist} / {len(hash1)}")
```

```
Collision: True
Hamming distance (avalanche): 0 / 256
```

Advantages of our Hash Function

Fast computation: The steps involve simple block parsing and discrete movements on a small 4x4 grid with quick edge weight updates — all these operations run efficiently, allowing fast hash computation, even long messages approximately take the same amount of time



	Message	Length
0	A	1
1	Quantum hashing provides a way to compress cla...	626
2	V. E. Schwab's The Invisible Life of Addie LaR...	3230
3	Leigh Bardugo's Six of Crows (2015) is a fast-...	6020
4	Cinder by Marissa Meyer is a captivating, futu...	7423

	Spectral Hash	Time (s)
0	[0.017014, 0.020354, 0.02523, 0.014687]	6.368981
1	[0.013278, 0.024679, 0.02134, 0.03218]	5.232902
2	[0.017022, 0.01564, 0.062873, 0.017022]	7.039819
3	[0.018408, 0.015955, 0.06954, 0.017715]	6.270398
4	[0.016706, 0.020387, 0.057597, 0.014931]	7.486793

Note: Regardless of the message length, the hash function requires the same computation time since every message is represented as a fixed 4×4 grid.

Advantages of our Hash Function

Fixed output size: After constructing the weighted graph from the 4×4 grid, we compute its adjacency and degree matrices to obtain the graph Laplacian. The spectral fingerprint derived from this Laplacian then serves as the fixed-length (256-bit) hash.

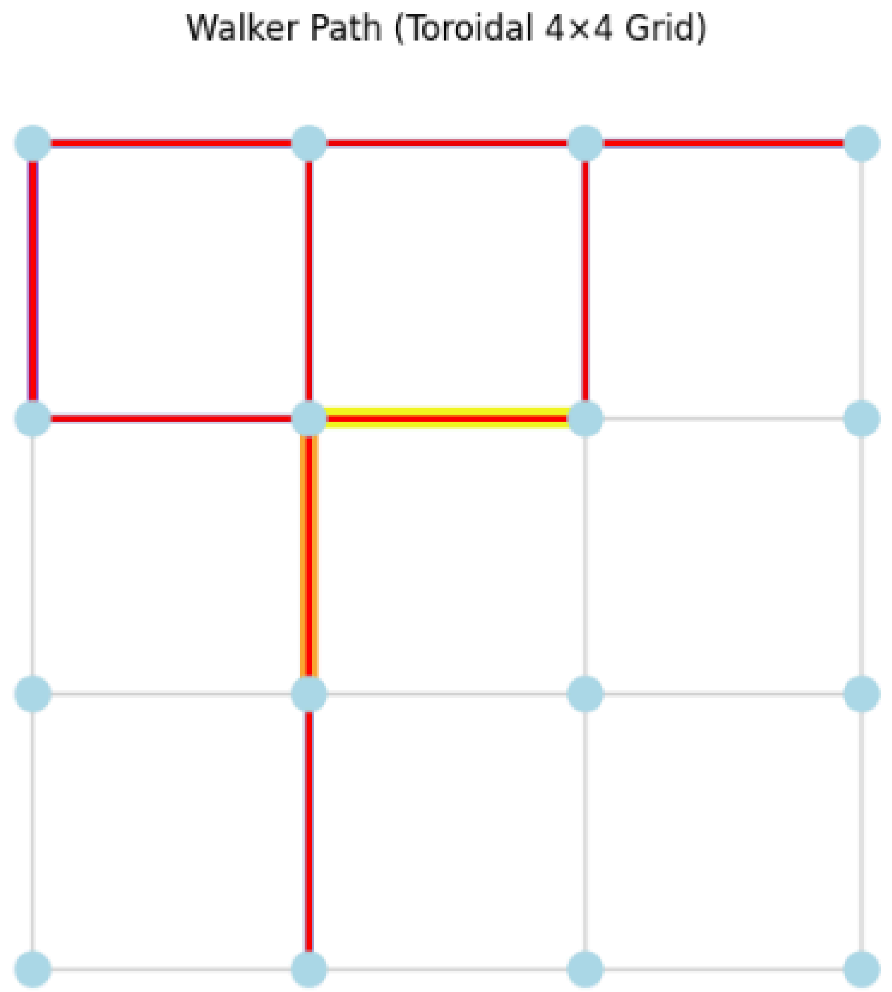
Pre-image resistance: Spectral fingerprinting transforms the adjacency matrix into a spectral signature; recovering the original message from this complex transformation and weighted graph is computationally difficult.

Uniformity: Because the walk paths and edge weights depend sensitively on input blocks, the resultant spectral fingerprints tend to distribute across the space evenly, helping avoid clustering and making the hash output appear random.

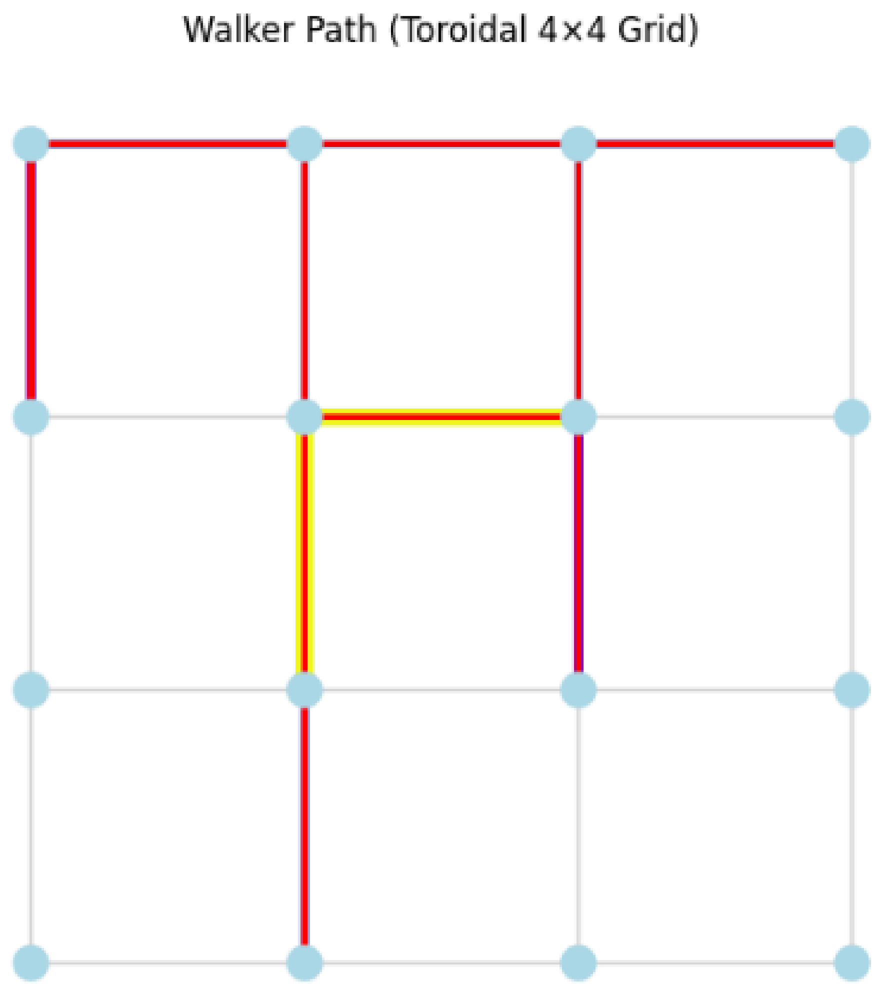
Advantages of our Hash Function

Collision resistance: Even small input changes alter the block sequences controlling walker directions, which changes the walk path and edge weights significantly, producing a substantially different adjacency matrix and spectral fingerprint, making collisions unlikely.

```
sf1 = spectral_fingerprint("hello")
hash1 = convert_spectral_fingerprint_into_hash(sf1)
print(hash1)
```




```
sf2 = spectral_fingerprint("hella")
hash2 = convert_spectral_fingerprint_into_hash(sf2)
print(hash2)
```




```
# Test 1: Collision resistance
print(f"Collision: {hash1 == hash2}")
```

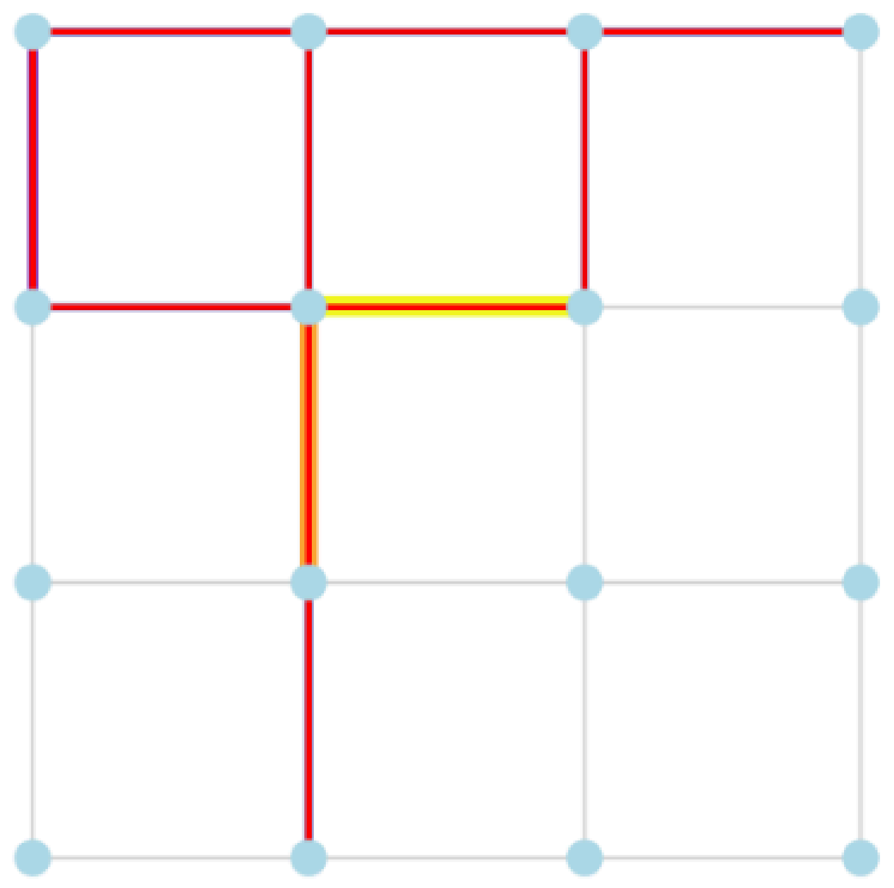
```
Collision: False
```

Advantages of our Hash Function

Avalanche effect: Since each 2-bit block change affects walker movement and edge weights, minor input variations lead to very different weighted graphs and corresponding spectral fingerprints, inducing large output differences from small input changes.

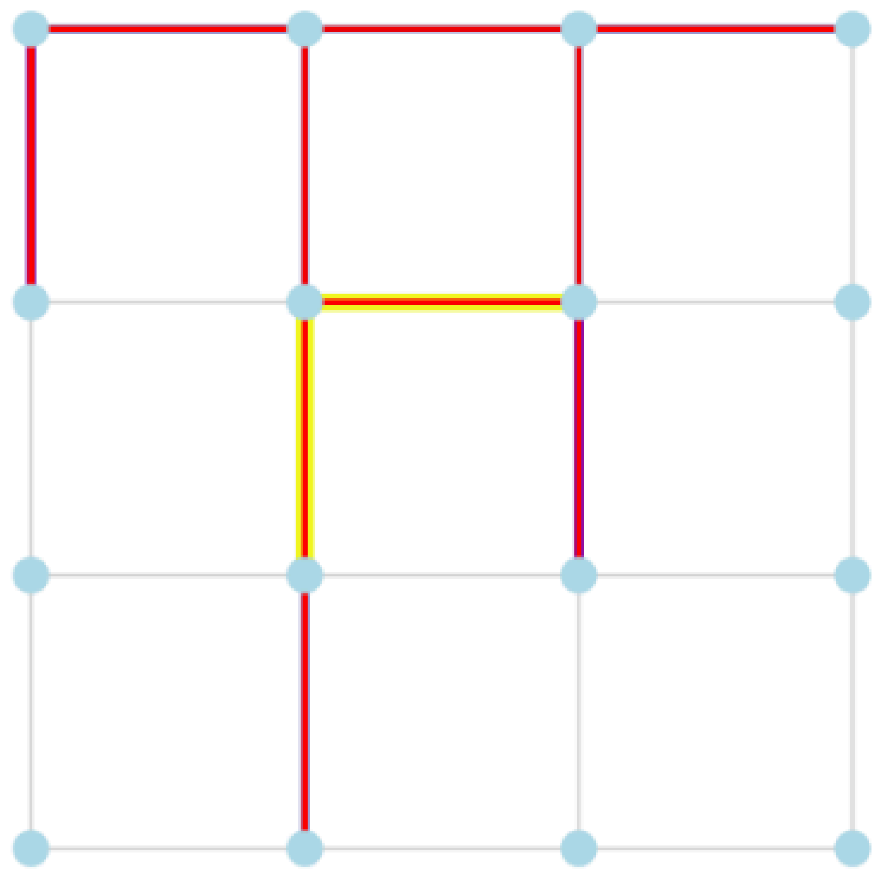
```
sf1 = spectral_fingerprint("hello")
hash1 = convert_spectral_fingerprint_into_hash(sf1)
print(hash1)
```

Walker Path (Toroidal 4×4 Grid)



```
sf2 = spectral_fingerprint("hella")
hash2 = convert_spectral_fingerprint_into_hash(sf2)
print(hash2)
```

Walker Path (Toroidal 4×4 Grid)



```
def hamming_distance(a, b):
    return sum(x != y for x, y in zip(a, b))

# Test 2: Avalanche effect
dist = hamming_distance(hash1, hash2)
print(f"Hamming distance (avalanche): {dist} / {len(hash1)}")
```

```
Hamming distance (avalanche): 108 / 256
```

Birthday Attack

The birthday attack is based on the birthday paradox in probability. In hashing, we can find two different inputs with the same hash output (collision) much faster than trying all possibilities while using brute force. In our hash function, we tried simulating this for 100 trials:-

Code:

```
import random

def test_birthday_collision(max_trials=100):
    hashes = set()

    for i in range(max_trials):
        # Generate a random input message
        message = str(random.getrandbits(256))

        # --- Use your custom spectral fingerprint hash ---
        sf = spectral_fingerprint(message)
        h = convert_spectral_fingerprint_into_hash(sf)

        # --- Check for collision ---
        if h in hashes:
            print(f"Collision found after {i} hashes!")
            return i
        hashes.add(h)

    print(f"No collision found after {max_trials} trials.")
    return None

test_birthday_collision()
```

Output:

```
No collision found after 100 trials.
```

Why QGH_256?

Quantum computing represents a paradigm shift in computational science, leveraging the principles of quantum mechanics—such as superposition and entanglement to solve problems that are computationally infeasible for classical systems [1], [2], [3], [4]. This emerging technology introduces fundamentally new ways of processing information, enabling certain algorithms to outperform their classical counterparts by orders of magnitude. Among the most prominent examples are Shor's algorithm for factoring large integers and computing discrete logarithms [5], and Grover's algorithm for unstructured search [6], both of which have direct implications for the foundational structures of modern cryptography. The exponential speedup offered by Shor's algorithm threatens to render widely used public-key encryption schemes—such as RSA and elliptic curve cryptography (ECC) obsolete [7], while Grover's quadratic speedup reduces the effective security of symmetric-key algorithms, including AES [8], [9]. These vulnerabilities could compromise the confidentiality, integrity, and authenticity of sensitive data across financial, governmental, military, and civilian digital infrastructures.

reference :- https://www.sciencedirect.com/science/article/abs/pii/S0045790625005920

- Quantum hash resists Grover's algorithm via randomized coin-flip pairings, with further research we could also bring in keys which lock specific pairings.
- Spectral fingerprinting of the graph Laplacian strengthens pre-image resistance.
- Shor's algorithm does not threaten this hash (only affects RSA/ECC).
- No classical post-processing needed, keeping the scheme efficient.

Decoding complexity

Spectral fingerprint to weighted graph → Weighted graph to bit string → 00 is up or down or left or right ?

Nearly impossible | Too many possibilities | Again 4 more possibilities

Reference

1) https://www.sciencedirect.com/science/article/abs/pii/S0045790625005920
2) https://medium.com/@belal.db/quantum-phase-estimation-the-math-behind-the-circuit-59bc45e1e339
3) https://arxiv.org/abs/2408.03672
4) https://arxiv.org/abs/2510.01918
5) https://www.researchgate.net/publication/388349534_Suzuki-Trotter_Decomposition_A_Step-by-Step_Theoretical_proof_of_the_formulae

For our code

Google Colab:
https://colab.research.google.com/drive/1CIxOT788CyK7fglTeVPwVlX63sF33GrD?usp=sharing

Github:
https://github.com/pranavishvar/QGH_256.git

THANK YOU